**EML 5526 Final Report**

Project B – 2D Heat Transfer

By

Mackenzie Ott 22391351

Table of Contents

[Table of Equations 3](#_Toc437330259)

[Table of Figures 3](#_Toc437330260)

[Overview of Problem 4](#_Toc437330261)

[Theoretical Work 4](#_Toc437330262)

[Details of Elements 4](#_Toc437330263)

[Derivation of the discretized weak form 4](#_Toc437330264)

[Code Outline 6](#_Toc437330265)

[Main Function 6](#_Toc437330266)

[Map Function 10](#_Toc437330267)

[Jacobian Function 10](#_Toc437330268)

[Element Stiffness 11](#_Toc437330269)

[Temperature Location Function 11](#_Toc437330270)

[ColorLine3 Function 12](#_Toc437330271)

[Heat Flux Function 12](#_Toc437330272)

[Analysis of Code using Simple Problem 12](#_Toc437330273)

[Conclusion 16](#_Toc437330274)

[Limitations and Potential Improvements of Program 16](#_Toc437330275)

[Appendix 17](#_Toc437330276)

[Software Flow Chart 17](#_Toc437330277)

# Table of Equations

[Equation 1 Equilibrium Equation 4](#_Toc437106315)

[Equation 2 Fourier's Law 4](#_Toc437106316)

[Equation 3 Governing Equation 4](#_Toc437106317)

[Equation 4 Simplified Governing Equation 4](#_Toc437106318)

[Equation 5 Weighted Residual Method 4](#_Toc437106319)

[Equation 6 Green's Theorem 5](#_Toc437106320)

[Equation 7 Final Form of Weak Form 5](#_Toc437106321)

[Equation 8 Temperature Function 5](#_Toc437106322)

[Equation 9 Mapping Function 5](#_Toc437106323)

[Equation 10 Shape Function for a 4-node Quaderateral 5](#_Toc437106324)

[Equation 11 Element Stiffness Matrix 6](#_Toc437106325)

[Equation 12 Jacobian Matrix 11](#_Toc437106326)

# Table of Figures

[Figure 1 Pull Data Code Section 6](#_Toc437106331)

[Figure 2 Calculate Element Stiffness 7](file:///C:\Users\khaji_000\Documents\Final%20Report.docx#_Toc437106332)

[Figure 3 Set RHS 8](file:///C:\Users\khaji_000\Documents\Final%20Report.docx#_Toc437106333)

[Figure 4 Compute Temperature Nodes and Heat Flux 9](#_Toc437106334)

[Figure 5 Write Data 9](#_Toc437106335)

[Figure 6 2-D Color Map 10](#_Toc437106336)

[Figure 7 Map Function 10](#_Toc437106337)

[Figure 8 Jacobian Function 10](#_Toc437106338)

[Figure 9 Element Stiffness Function 11](#_Toc437106339)

[Figure 10 Temperature Location Function 11](#_Toc437106340)

[Figure 11 ColorLine3 Function 12](#_Toc437106341)

[Figure 12 Problem Statement 13](#_Toc437106342)

[Figure 13 Boundary Mesh Diagram 13](#_Toc437106343)

[Figure 14 2-D Heat Color Map – 2 Elements 15](#_Toc437106344)

[Figure 15 SolidWorks Temperature Output 15](#_Toc437106345)

[Figure 16 MATLAB Code Heat Flux Result 16](#_Toc437106346)

[Figure 17 Solidworks Heatflux Result 16](#_Toc437106347)

# Overview of Problem

The scope of this assignment was to implement a finite element using any programming language of my choice. I chose to implement a 2-D heat transfer finite element. The boundary mesh and the material properties are read from an outside file (in this case an Excel file) and the output data is written to a different file (a text document in this case for the raw data and .BMP for the graph).

# Theoretical Work

## Details of Element

The element used for this project was a 4-Node quadrilateral. An 8-node quadrilateral could implemented, but would be much more complicated and was not needed for the simple problems being analyzed. The nodes are labeled as shown below:

4

2

1

3

Figure 1 Labeling of Nodes

## Derivation of the Discretized Weak Form

The governing equation of 2-D heat transfer can be found by substituting Fourier’s Law into the Equilibrium equation. The equations are shown below.

Equation 1 Equilibrium Equation

Equation 2 Fourier's Law

Equation 3 Governing Equation

Since *kxy* and *kyx* are considered to be 0 and *kxx* = *kyy* = *k* for an isotropic element, the governing equation can be shown using vector notation as:

Equation 4 Simplified Governing Equation

The next step is to convert this into a weak form to create a integral equation.

Equation 5 Weighted Residual Method

Since T and k are continuous, differential scalar fields defined over a volume V, Green’s theorem can be used.

Equation 6 Green's Theorem

When applied to Equation 5, the resulting equation becomes

Equation 7 Final Form of Weak Form

To turn this into a usable form for a FEM solver, Equation 7 needs to be discretized. To do this, the element needs to be changed into a perfect isoparametric element to make integrating easier. The temperature can be interpolated using shape functions as thus:

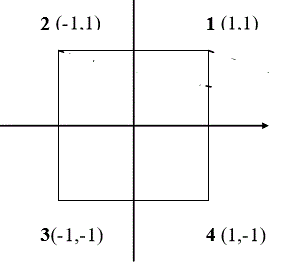
Equation 8 Temperature Function

Where s and t are the coordinates on the parametric element. Similarly, the mapping between the parametric space and the real space is defined as

Equation 9 Mapping Function

Since at each of the 4 corner nodes, only one shape function be equal to 1, the four shape functions can be written as thus.

Equation 10 Shape Function for a 4-node Quaderateral



Equation 11 Layout of Shape Functions

When applied to Equation 7, the equation when written in matrix notation becomes:

Equation 12 Final Equation for Element

Equation 13 Final Equation for Global

Equation 14 Element Stiffness Matrix

# Code Outline

### Main Function

The code path is outlined in the Appendix under the heading Code Flowchart. The individual sections of the code are summarized below.

#### Initialization and Pull Data

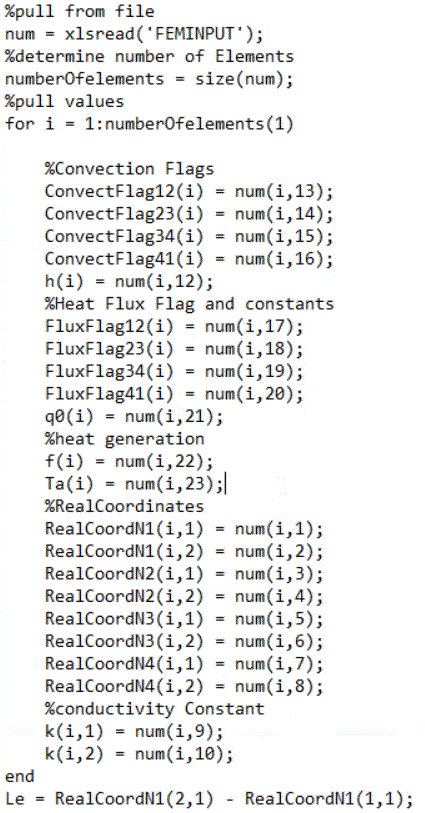
**

Figure 2 Pull Data Code Section

The program is initialized by reading the boundary mesh and material properties from the FEMINPUT Excel file (which can be found in the containing folder) and assign values to variables that will be used throughout the program. Each element is listed as its own row on the Excel file and contains its own boundary flags, material properties, etc.

#### Create Global Element Stiffness Matrix

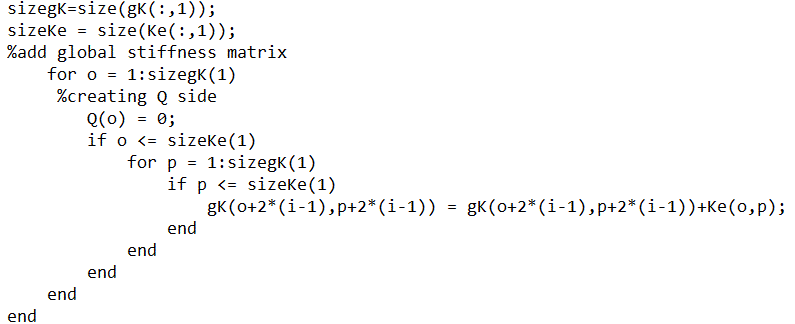
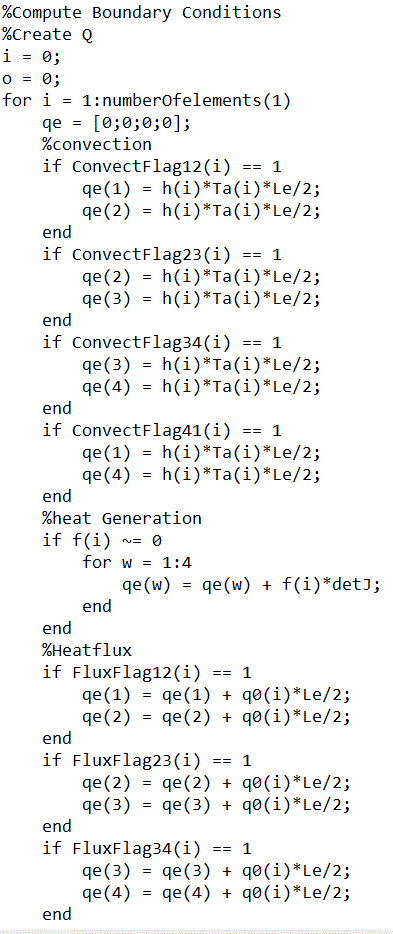


Figure 3 Calculate Element Stiffness

This part of the code calls the Jacobian function and the Element Stiffness function for each element, creating an element stiffness matrix, adds the effects of convection on the stiffness matrix if a convection flag is set. , and then adds the element stiffness matrix to the global stiffness matrix.

#### Create R.H.S and Compute Boundary Conditions



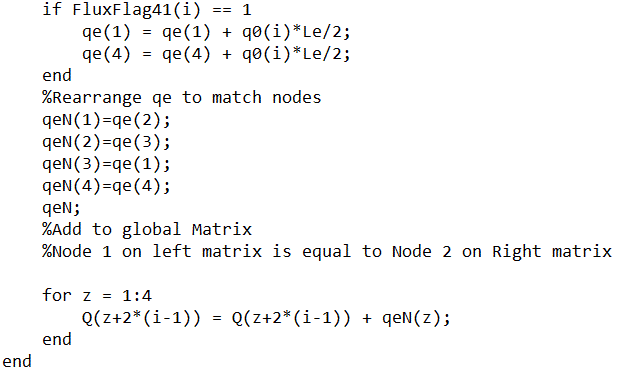
To generate the RHS of the equation, the program checks for flags set in the mesh. If a flag is set, the programs add the effects of the flag to the RHS, and then rearranges it to fit the element mesh outlined previously in Figure 1. The current flags it can read are ones for heat convection, heat flux, and heat generation. Convection and flux are flagged per element side, while generation is flagged per element. The element RHS is then added to the global RHS.

Figure 4 Set RHS

#### Compute Temperature Nodes and Heat Flux and Create Temperature Mesh

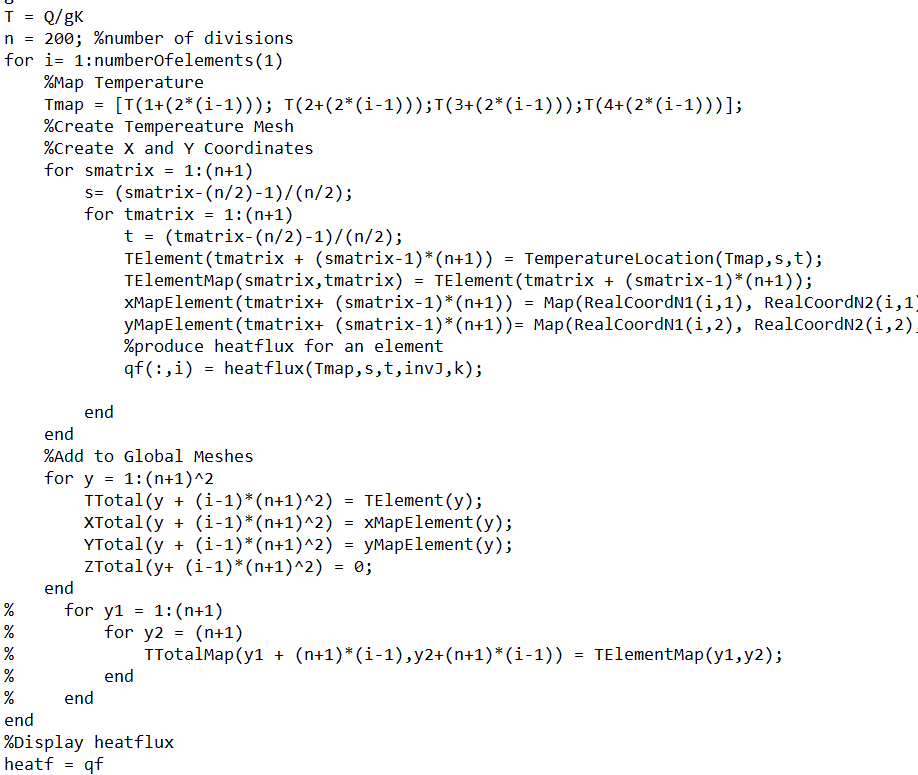


Figure 5 Compute Temperature Nodes and Heat Flux

This loop calculates the global temperature nodes using Equation 13, then produces a temperature reading for each x- and y-coordinate by calling the Temperature Location function. It also add as a z-coordinate with a value of 0 for later use in a plotting function. It also calculates the heat flux for each element involved using Equation 2.

#### Write Data

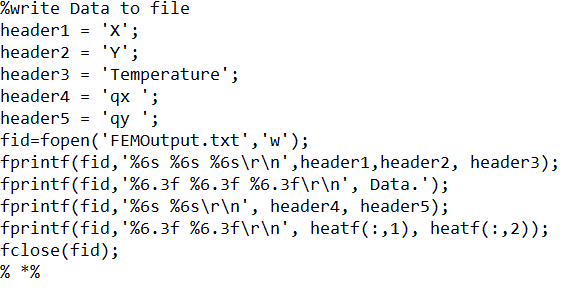


Figure 6 Write Data

This section of the code opens a text file called FEMOutput, and writes the temperature mesh in a x,y,T format. It then writes the heat flux of each element present in the problem.

#### Create 2-D Color Map

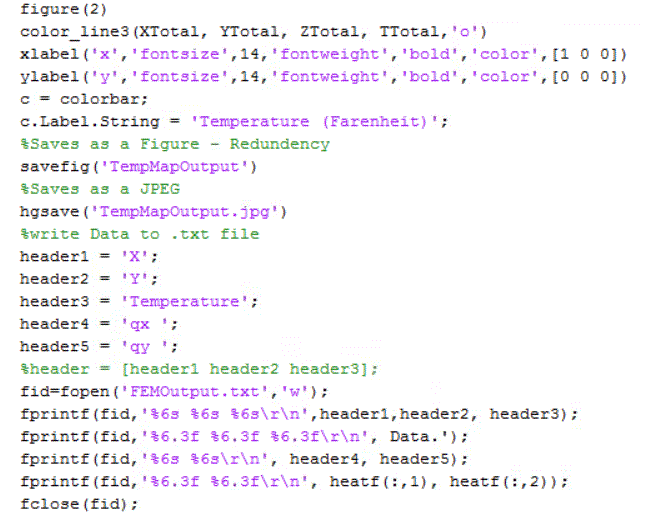


Figure 7 2-D Color Map

This section of the code generates the 2-D heat map of the problem by calling color\_line3, then labels the map appropriately. ZTotal is a dummy variable sized to match XTotal, YTotal, and TTotal, but every value is set to 0 in order to create a 2-D plot instead of a 3-D one. The program then write the plot to both a .fig file and a .jpg file called TempMapOutput.

### Map Function

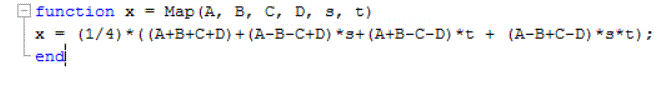
****

Figure 8 Map Function

This function takes in the real node x-, or y-coordinates (shown by A, B, C, and D) and the s- and t- coordinates of the desired point, and outputs the real coordinates for the s- and t-coordinates. The equation implemented in this function is Equation 9.

### Jacobian Function

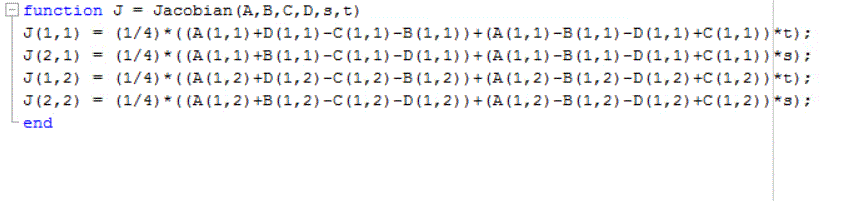


Figure 9 Jacobian Function

This function computes the Jacobian based off of the real element coordinates and the s and t coordinates desired. Since the element is assumed to be a perfect quadrilaterals, s and t are considered to be 0 for the simple problem analyzed in this paper.

Equation 15 Jacobian Matrix

### Element Stiffness Function

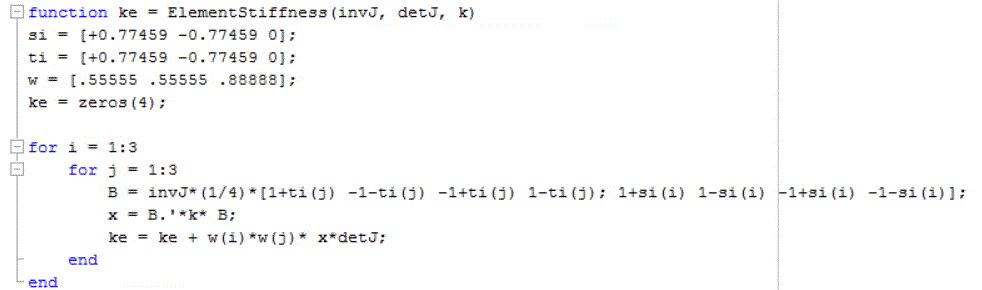


Figure 10 Element Stiffness Function

This function constructs the stiffness matrix for an element after receiving the inverse of the Jacobian matrix, the determinant of the Jacobian matrix, and the conductivity constant k of the element. To calculate the integral, a third order Gauss quadrature is used. Theoretically, a second order Gauss quadrature could be used, but as noted later in the simple problem analysis, it was actually insufficiently precise to reach the ideal stiffness matrix.

### Temperature Location Function

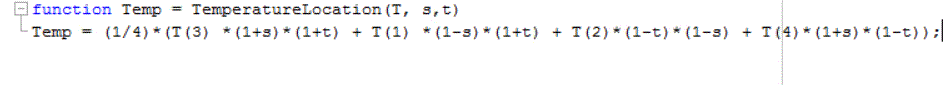


Figure 11 Temperature Location Function

This function outputs the temperature at a given point within an element from the four temperature nodes in the element and the s- and t- coordinates in the element.

### ColorLine3 Function

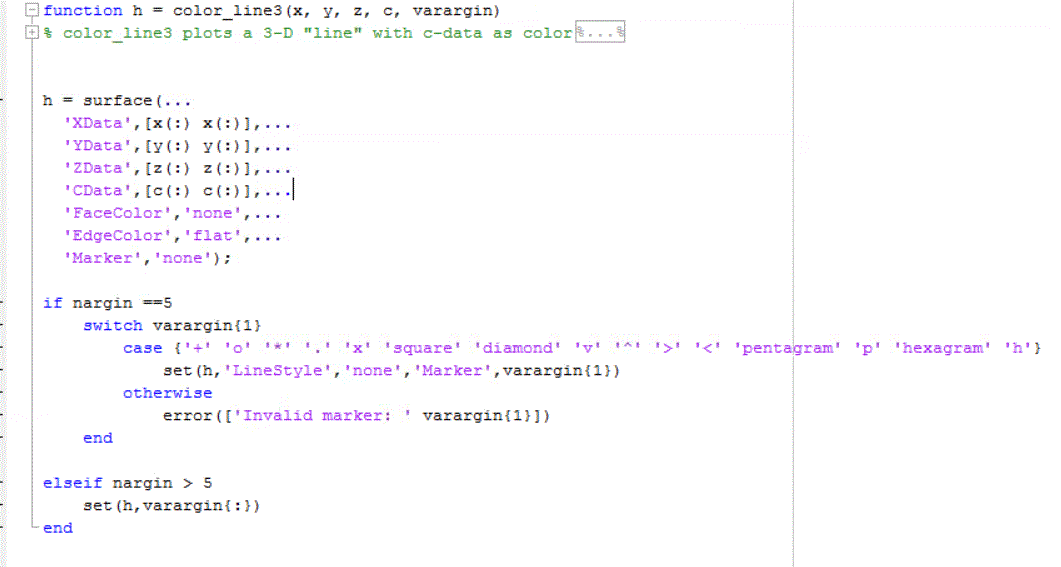


Figure 12 ColorLine3 Function

This function creates a colored 3d line plot using X, Y, Z, and T inputs. It can also create a colored scatter plot. In this project, the function is used to create the 2-D temperature map shown in the analysis section by setting all values of Z to 0.

### Heat Flux Function

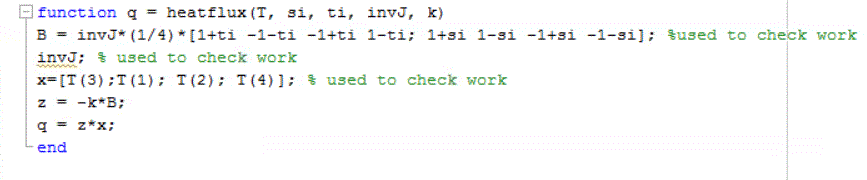


Figure 13 Heat Flux Function

This function calculates the average heat flux through an element from the inputs of the temperature nodes, s- and t- coordinates, inverse of the Jacobian, and k. It then produces a 2X1 matrix containing qx and qy.

# Analysis of Code using Simple Problem

The simple problem used to confirm the success of the code was the example from the class notes 4\_HT2D. The problem is shown below in Figure 12.

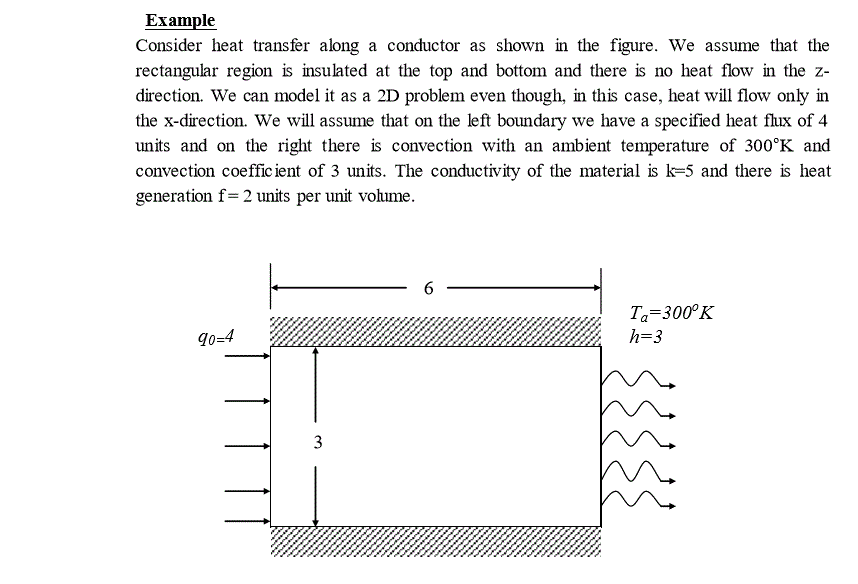


Figure 14 Problem Statement

For this problem, the mesh boundary will consist of two elements, each 3 units by 3 units in size, with the boundary conditions as shown below.

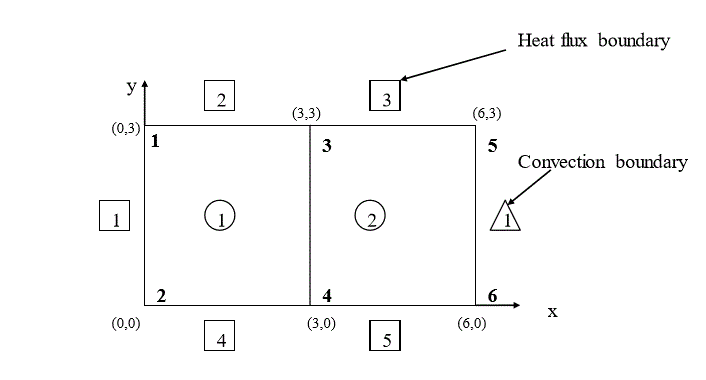
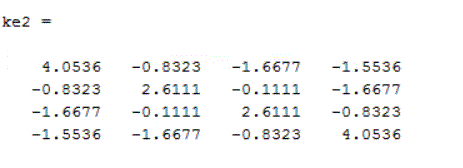
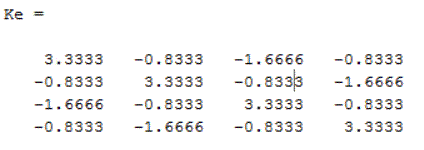


Figure 15 Boundary Mesh Diagram

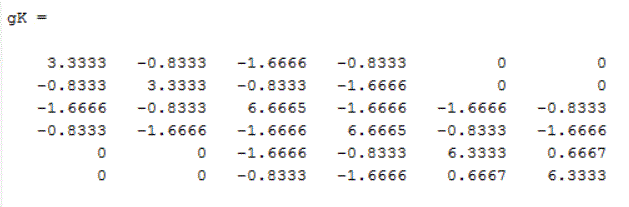
Due to the symmetry of the problem and the perfect rectangle of the element, the element stiffness matrix will be the same between elements. Using a second-order Gauss Quadrature, the element stiffness matrix was found to be:



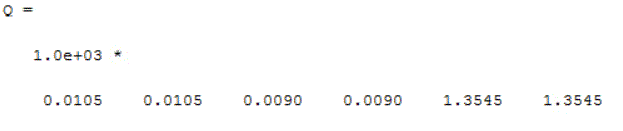
However, a third-order Gauss Quadrature found the element stiffness matrix to be:



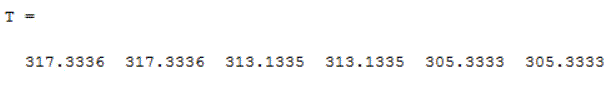
This is more in line with the matrix found through integration. With the effects of convection added in, the global stiffness matrix becomes:



The next step is to develop the right side of the equation. With the effects of heat flux, heat convection, and heat generation, the right hand side of the equation becomes:



Which matches up with the values obtained in the notes. The T-node values calculated are shown below



Using the temperature values calculated above, a 2-d color map is produced:

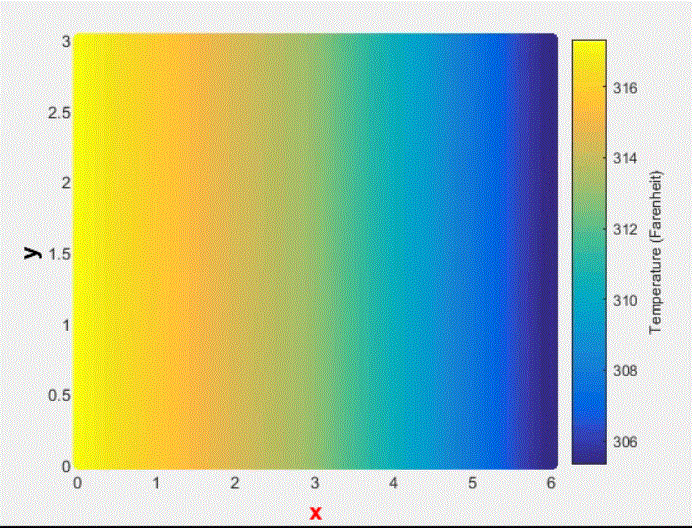


Figure 16 2-D Heat Color Map – 2 Elements

This was compared to the SolidWorks FEM Analysis, which is shown below.

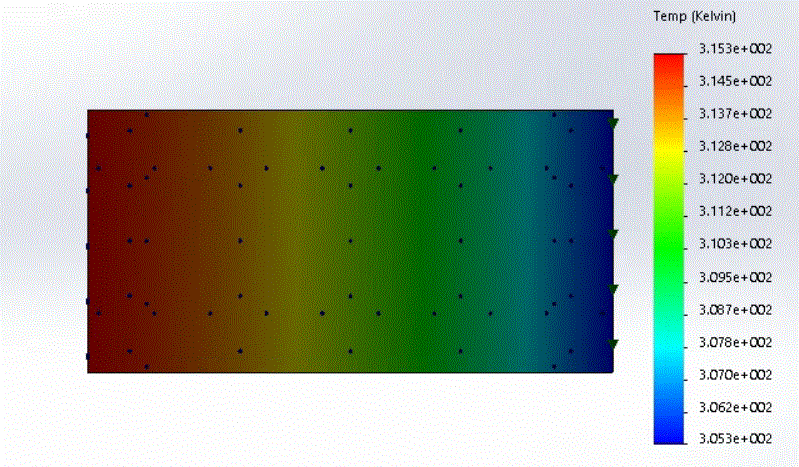


Figure 17 SolidWorks Temperature Output

The heat flux for each of the elements is shown below. The numbers makes sense since there was no heat change from top to bottom, so qy is zero. The SolidWorks plot of the heat flux in the x-direction is also shown below. If there were more elements in this analysis, the SolidWorks plot and the MATLAB code would match up better. As shown in Figure 17 below, the qx found is the average of the heat flux across each element.

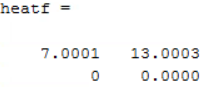


Figure 18 MATLAB Code Heat Flux Result

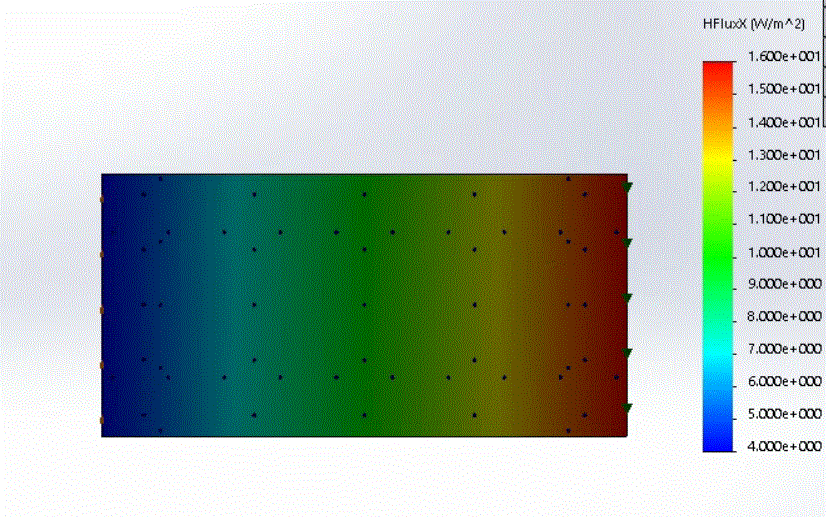


Figure 19 SolidWorks Heatflux Result

# Conclusion

## Limitations and Potential Improvements of Program

The program currently operates at its best when it is doing perfect rectangular real elements. Misshaped real elements cause the Jacobian to become dependent on the s- and t- coordinates, as opposed to the much easier constant Jacobian from a perfect rectangular element. It also cannot do multiple rows of elements. Potential future improvements to this code could include adding the feature of multiple rows to the program, but would double or even triple the size of the code due to the complexity of selecting where an element stiffness matrix fits into the global matrices based on its element location.

# Appendix

## Software Flow Chart

