Solving the 2D Heat Equation using Finite Element Analysis

Mahmoud Ayoub

250874659

mayoub6@uwo.ca

April 30th, 2019

Applied Math 4613B

Dr. Allan MacIsaac

Abstract

The purpose of this paper is to solve the 2D heat equation on a general polygonal domain using finite element analysis. To do this, GMSH was used to create a polygonal mesh which was then read in MATLAB and converted into a file that could be visualized on VisIt. It was found that the code ran with minimal error and produced values that are consistent with theory. This is important to understanding a key method to solving and visualizing partial differential equations in nature.

1 Introduction

The world around us is a mysterious place that requires more than just intuition to understand. In the late 1600's calculus was invented, and this allowed us to describe the physical phenomena that occurs in our world with extreme accuracy through differential equations. These equations however only had analytical solutions that were derived through symmetries of the problem. It wasn't up until Euler developed his own method for numerically approximating ordinary differential equations for us to develop more rigorous techniques. In the mid 1900's, the finite element method was created alongside the finite difference method, then allowing us to numerically solve partial differential equations.

While these methods are the still the same and haven't been drastically modified since their invention, they were extremely tedious, and some problems would take months or years to solve by hand. It was only recently in the last couple of decades have we been able to instantaneously solve these problems on our high-speed computers. With finite element method, we've been able to solve many different types of problems in structural, aeronautical, and mechanical engineering. It is especially useful for visualizing bending and twisting in beams, indicating stresses and displacements.

In our case, we've developed a program to solve the 2D heat equation on a general polygonal domain using triangles. The heat transferred from one boundary to the other is consistent with what we've calculated by hand, as see in the next section.

2 Theory

To gain a better understanding of the situation, we must ask ourselves, “What exactly is the finite element method and how do we use it to solve PDE’s?” First off, FEM is nothing more than breaking down our PDE into a linear algebra problem. This linear algebra problem is then solved by using a system of equations. Of course, it’s not all that simple and there are some rigorous derivations to go through. Let’s start off with the general heat equation (1) that states

$$k\nabla^2 u = -f \tag{1}$$

We want to solve this equation on a general two-dimensional polygonal domain. A good place to start would be to start meshing up our domain into a bunch of triangles. One may use squares/rectangles, but these impose some unwanted symmetries that may end up messing with our solution. To do this, lets start by developing the Lagrange basis function.

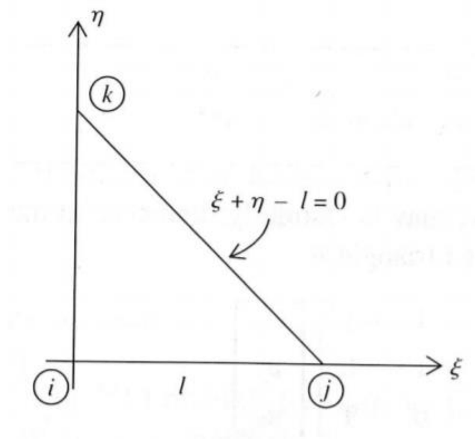

Figure 1: The 3-node master triangle we’ll be using for our analysis

We start off by setting our linear functional $\varphi = a + b\zeta + c\eta$, since $\varphi(0,0) = a = 1$, $\varphi(1,0) = a + b = 1$, $b = -1$, and $\varphi(0,1) = a + c = 1$, $c = -1$. So, our Lagrange basis function becomes,

$$\begin{aligned} \varphi_1 &= 1 - \zeta - \eta \\ \varphi_2 &= \zeta \\ \varphi_3 &= \eta \end{aligned} \tag{2}$$

Next, we need to formulate our Jacobian matrix. This is so we can switch from the global to our local coordinate system whenever we please, as we’re going to have a mesh made from many triangles that take on many different nodes and may not have the same exact shape throughout. Also, since out 3 nodes aren't evenly spaced out apart, we may use gaussian points for an increase in accuracy. We know by the definition of the Jacobian, and using the gaussian point,

$$J(\zeta_m,\eta_m) = \begin{matrix} \frac{\partial \chi}{\partial \zeta_m} & \frac{\partial \chi}{\partial \eta_m} \\ \frac{\partial \Upsilon}{\partial \zeta_m} & \frac{\partial \Upsilon}{\partial \eta_m} \end{matrix} \qquad (3) \quad , \quad \chi = \sum \chi_m \, \Phi_m \qquad (4)$$

Taking the inverse and then the determinant we find equations (5) and (6)

$$J(\zeta_m,\eta_m)^{-1} = \frac{1}{\frac{\partial \chi}{\partial \zeta_m}\frac{\partial \Upsilon}{\partial \eta_m} - \frac{\partial \chi}{\partial \eta_m}\frac{\partial \Upsilon}{\partial \zeta_m}} \begin{matrix} \frac{\partial \Upsilon}{\partial \eta_m} & -\frac{\partial \chi}{\partial \eta_m} \\ -\frac{\partial \Upsilon}{\partial \zeta_m} & \frac{\partial \chi}{\partial \zeta_m} \end{matrix} \quad , \quad \det\big(J(\zeta_m,\eta_m)\big) = \frac{\partial \chi}{\partial \zeta_m}\frac{\partial \Upsilon}{\partial \eta_m} - \frac{\partial \chi}{\partial \eta_m}\frac{\partial \Upsilon}{\partial \zeta_m}$$

We can use the inverse to switch from our master triangle to our actual physical triangle,

$$\begin{matrix} \frac{\partial \Phi}{\partial x} \\ \frac{\partial \Phi}{\partial y} \end{matrix} = J\big(\zeta_m,\eta_m\big)^{-1} \begin{matrix} \frac{\partial \Phi}{\partial \zeta} \\ \frac{\partial \Phi}{\partial \eta} \end{matrix} \qquad (7)$$

Basically, what we're doing is mapping our master triangle to the boundary that will be made up of different shaped triangles.

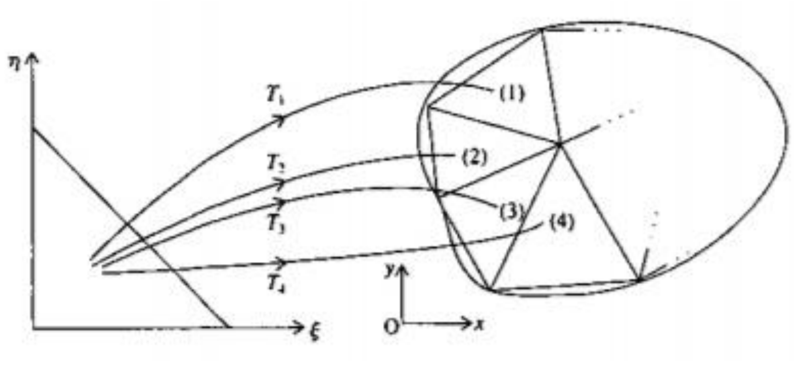

Figure 2: Successive mappings onto the boundary using the Jacobian

Now we use something called the Galerkin method to change this equation into variational form, and from this variational form we can change it into our linear algebra problem. From this, we can get our stiffness matrix for the problem. After multiplying both sides by the weight ω and integrating both sides,

$$\int_{\Omega(x,y,z)} \omega \left(\frac{\partial}{\partial x_i} \left(k \frac{\partial u}{\partial x_j} \right) + f \right) d\Omega = 0 \tag{8}$$

$$\Rightarrow \qquad \int_{\Omega} k \left(\frac{\partial \omega}{\partial x_i} \left(\frac{\partial u}{\partial x_j} \right) \right) d\Omega \ = \ \int_{\partial\Omega} \omega \left(k \frac{\partial u}{\partial x_j} \, \widehat{\boldsymbol{n}_j} \right) d\Omega + \ \int_{\Omega} \omega(f) \ d\Omega \tag{9}$$

Since the last term is zero, and ω = Φ$_i$, we can now write to find out stiffness matrix,

$$\Rightarrow \qquad K \ = \sum_{gaussian\ \#=1}^{m} C(x(\zeta_m), y(\eta_m)) \ \frac{\partial\Omega}{\partial x_i} \frac{\partial\Omega}{\partial x_j} * W_k * |\det(J(\zeta_m, \eta_m))| \tag{10}$$

Now we can finally solve our linear algebra problem given our boundary conditions!

3 Results and Discussion

To begin with, we used GMSH to create a square boundary that we used as our 2D mesh. Breaking it up into 4 different elements and 5 different nodes as shown in figure 3. This was the trial shape to make sure our code ran successfully. Also, we did some calculations by hand to make sure that our final stiffness matrix was identical to what we got through our code. Finally, the mesh was replotted on VisIt where it can be better visualized. We had to add a function in our code so that we could read our .dat file that we later create at the end. Now setting up the code was a bit tedious but going through it step by step will insure a good understanding of what's going on. Also, to ensure you have a correct .gmsh file, make sure to add the physical contours and surface! This makes the tags that we look for in the code that correspond to our boundaries.

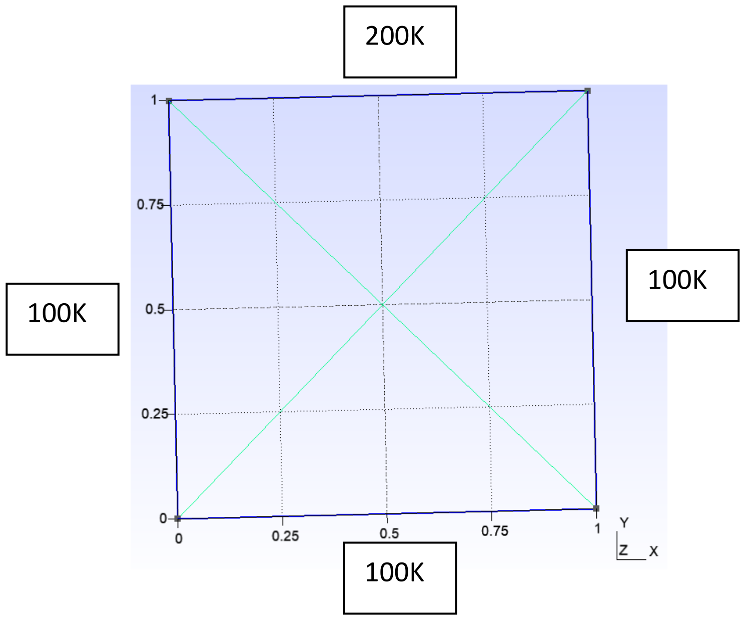

Figure 3: Our reference boundary to verify that our code works

Outline of the code

First, we began by making a function() to read our GMSH file, read_gmsh_2d(), which took in the input of the file and an empty structure that we filled up with the GMSH file information, such as the number of nodes, number of elements, surfaces, boundaries, etc. We then coded our function() for the Lagrange basis function derivatives which was needed to code the function() for the Jacobian. The Jacobian was then used to code the function() to change from the global coordinate system to the local one. After doing so, the function() for the stiffness matrix was coded from our equation (9), and allowed us to then create the function() which calculates the final stiffness matrix from all the elements. Then the function() for the boundary conditions was made, and we needed to make sure we could satisfy that the specific formatting of the file we were given. In the formatting of the .msh file, we can see the number of nodes and elements corresponding to our boundary, and from this we could apply our boundary conditions. Finally, the project_FEM() function was made, which takes the file and the boundary conditions as the input. The file needs to exported in .msh format when in GMSH, and it has to be saved in Version 2 ASCII, or else the code won’t run. This is because GMSH came out with a new formatting for the meshes which we won’t be using.

Looking at our reference boundary, we have 5 nodes and 4 triangular elements. The 5th node lies exactly in the middle. After using our equations (2) – (7), we were able to derive the stiffness matrix by adding up the matrices of all 4 elements with the boundary conditions

$$K = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -1.5 & -1.5 & -1.5 & -1.5 & 6 \end{matrix}$$

Comparing this with what we get from MATLAB, it's the same! Moving forward, we can now mesh our boundary into more elements now that we know it works for our reference boundary.

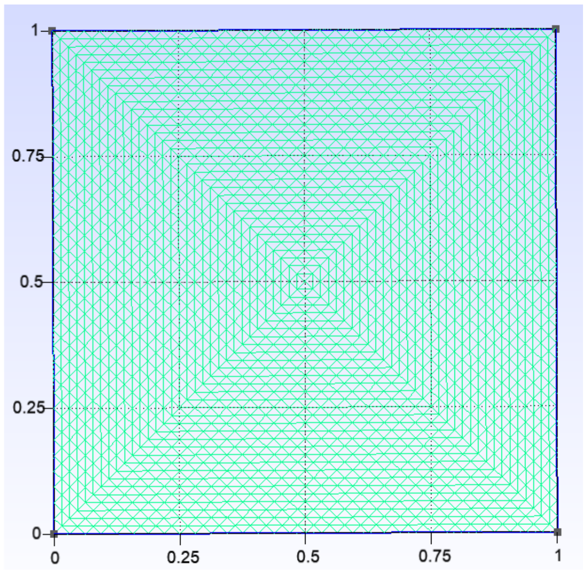

Figure 4: Our refined mesh with a large number of elements and nodes

Running our code now with the boundary conditions of 300K and 400K now, we can see the heat flux plotted in figure 5 as the temperature changes from one boundary to the other. Whether these values are correct is a different matter in numerical analysis, where we could find the errors that are produced from solving the linear algebra problem. However, we'll take our code for face value as we see that it matches with the simplest of meshes.

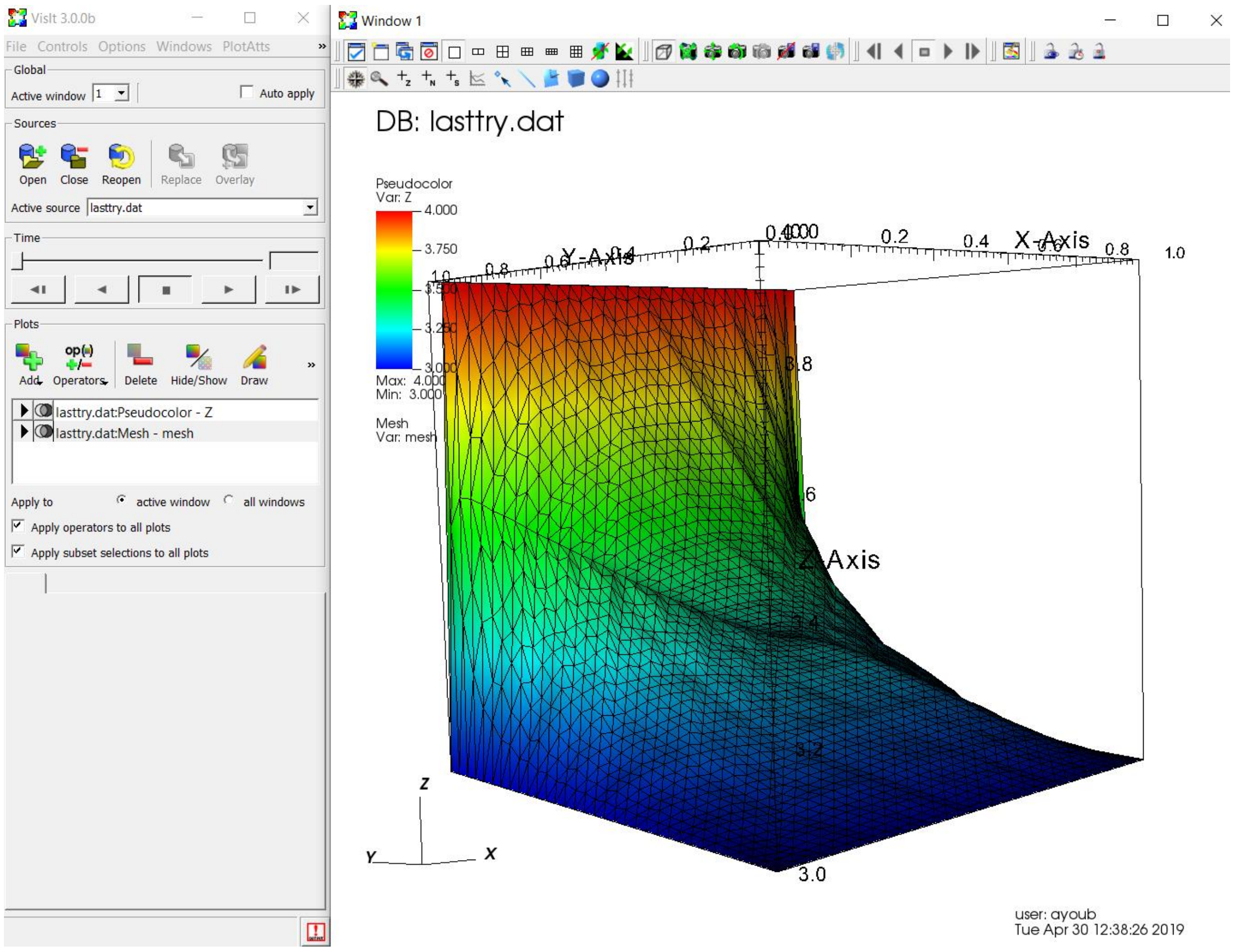

Figure 5: Our mesh with the temperature plotted in the z axis to visualize the heat flux from one boundary to the other

4 Conclusion

As we can see, the finite element method is a very powerful tool to solve differential equations on boundaries that we may not have solved using the finite difference method. The final solutions given from this code may not exact as it is not optimized, but it gives a good idea of how FEA should work given a general polygonal mesh. All in all, this project was a success and provided many insights on how we solve problems in math/physics using differential equations.

References

- https://drive.google.com/file/d/1gCeiX0Zcnk1H3vKIMmH1iHN0EQvDs6Jh/view - Murad Al Qurishee his own project on how to code the functions above
- Finite Elements A Gentle Introduction by David Henwood and Javier Bonet 1996