FINITE ELEMENT ANALYSIS

ARDEN RASMUSSEN

APRIL 16, 2019

Abstract. Finite Element Analysis (FEA) is the computational process prescribed by Finite Element Methods (FEM) in order to solve for numerical solutions to boundary value problems for partial differential equations. Finite Element Analysis is used in many specializations of mechanical engineering, and is used frequently in that discipline, and occasionally in other disciplines as well. This paper provides an introduction to the mathematics of Finite Element Methods, and why they are useful. Then, it provides and explains the simplistic implementation of Finite Element Analysis. The implementation provided with this paper is written in C++, and is commented for better readability of the code.

1. Introduction

Finite element methods is a computational process used for the determination of an approximate solution to a given partial differential equation, with given forcing term and boundary conditions. Finite element analysis is the practical implementation of finite element methods computationally on a physical problem in order to ascertain more insight into the physical system in question. We will first explain finite element methods, and provide a thorough introduction into the mathematics behind the methods. Then we will also provide an explanation of finite element analysis, demonstrating the mathematics of FEM with respect to an example problem, and develop a computational program that we will use to solve the partial differential equation using FEM.

Finite element methods is very useful, due to the fact that there are only a few situations where a solution to a partial differential equation can be found analytically, so we need methods to find a close approximation of the actual solution. Currently there are a handful of methods that can be used for this purpose, however finite element methods are currently the best methods that we know of. Meaning that the error between the analytical solution and the solution constructed by finite element analysis converges to zero faster than other methods.

The process that we utilize for constructing the finite element methods is called the Galerkin method, and is presented by [KH15]. We will closely follow the method outlined by their paper. For this introduction we will limit the scope to two dimensions, but the mathematics can be easily generalized to higher dimensions, and we will make note of the sections that would have to change for higher dimension problems.

We divide this paper into two major parts. Part 1 explains the process of FEM, and demonstrates the mathematics and the concepts behind the method, and how it can be used to solve the partial differential equations. Part 2 explains the process of FEA, and discusses the algorithms, data structures, and optimizations that can be done to construct a program to efficiently compute the solution utilizing the previously discussed FEM.

Part 1. Mathematics

2. Problem Statement

Throughout this paper and our sample code, we implement finite element methods using the general form of the heat equation as our differential equation to consider. We define the domain to be Ω such that $\Omega \in \mathbb{R}^2$ is a bounded 2D domain with boundary Γ.

The general form of the heat equation takes the form

$$\frac{\partial u}{\partial t} = Lu + f,$$

where L is an operator of the form

$$L = \sum_{\alpha,\beta=1}^{2} A^{\alpha\beta} \frac{\partial}{\partial x^\alpha} \frac{\partial}{\partial x^\beta} + \sum_{\alpha=1}^{2} B^\alpha \frac{\partial}{\partial x^\alpha} + C$$

and f is some forcing function. We will also enforce that A is a positive definite matrix. And where u is a function that takes the position and the time and returns a real $u(\vec{x}, t) : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$. Note that $\vec{x} \in \Omega$, and $t \in \mathbb{R}$.

Rewriting this expression into a single equation we can express our problem statement as

$$\frac{\partial u}{\partial t} = \sum_{\alpha,\beta=1}^{2} A^{ij} \frac{\partial}{\partial x^\alpha} \left[\frac{\partial u}{\partial x^\beta} \right] + \sum_{\gamma=1}^{2} B^\gamma \frac{\partial u}{\partial x^\gamma} + Cu + f \tag{2.1}$$

Note that for this process to function, we must also require some conditions. We first need the boundary conditions in order to determine any solution. Therefor we require that $u(\partial\Omega, t)$ is provided. Note that this could mean that the boundary is split into several sections, each one with different conditions, but the value of u at any point along the boundary must be known.

If considering the more general time dependent form of the equation we are also required to know initial conditions. That is to say that $u(\Omega, 0)$ must also be provided. Using the initial conditions and the boundary conditions, we are able to construct an approximation through the use of finite element methods.

Given A, B, C, f, and appropriate conditions, we must construct a solution for u. This is the key purpose of finite element methods, as for most circumstances there does not exists an analytical solution to this differential equation. Thus we must utilize finite element analysis in order to construct the best possible approximation of the solution to the differential equation, and then it is possible to use that approximation for further analysis.

3. Conditions

For our proposed problem statement, we will require some conditions in order to make it possible to determine a solution to the problem. The conditions that we will require are boundary conditions and if we are considering a time dependent system, we will also need an initial condition.

3.1. **Boundary Conditions.** We must prescribe boundary conditions on our problem statement, to ensure that there is a single solution. For now we will only proceed utilizing Dirichlet boundary conditions. It should be noted that other conditions are possible with this method, such as Neumann, and Newton boundary conditions. These other boundary conditions require more changes to the formulation of the problem later on and require extra modifications to the implementation. Because of the extra alterations that they would necessitate, we will only focus on the Dirichlet boundary conditions.

For the Dirichlet boundary conditions we must define the value of our solution at every point on the boundary. We define this as

$$\partial u(\vec{x}, t) \quad \vec{x} \in \partial\Omega.$$

We will use this notation to denote the function defining the values of the solution to the problem along the boundary at time t. This can be written as

$$u(\partial\Omega, t) \equiv \partial u(\partial\Omega, t).$$

It is important to note that there is no restrictions to the form of ∂u. The user defined function can take any form, and thus requiring the solution to also fit any form of boundary conditions.

3.2. **Initial Conditions.** If the problem being considered were to be time dependent, then it is also necessary to define an initial condition to the problem. That is to say that it is required to provide some function that defines the value of u at $t = 0$. We define this function as

$$u_0(\vec{x}) \quad \vec{x} \in \Omega.$$

We use this notation to denote the function defining the values of the solution to the problem, that is to say

$$u(\Omega, t = 0) \equiv u_0(\Omega).$$

Note that this function also has no restrictions other that it must be defined on the entirety of the domain Ω, although it need not be continuous on the domain.

If the problem being considered is not time dependent, than the initial condition can be ignored, and only the boundary conditions must be specified.

Using these prescribed conditions, both the boundary and the initial conditions, it is now possible to utilize finite element methods to attain an approximation of the solution to the problem statement.

4. Variational Formulation

In order to construction the variational formulation of our problem statement, we must first introduce the concept of a residual. The residual is the construction of a function R such that when our approximation is the exact solution then $R = 0$. Using the residual we now want to construct a function that minimizes the value of the residual. We will construct the residual of equation 2.1 as

$$R(\vec{x}, t) = \frac{\partial u}{\partial t} - Lu - f$$

Clearly by the construction of R, $R(\vec{x},t)=0$ if and only if u is the exact solution to the partial differential equation. Now we multiply the residual by some arbitrary test function, and integrate over our domain

$$\int_\Omega w(\vec{x})R(\vec{x},t)d\vec{x}.$$

For the exact solution u this will always be zero, because $R(\vec{x},t)=$ for all $\vec{x}\in\Omega$. Because of this we are able to select any test function w, and the weighted residual will still be zero, given the exact solution. However, we will restrict our test functions to ones that are zero along the boundaries $\partial\Omega$, this will significantly simply our computations in the future.

We now expand the weighted residual to obtain

$$\int_\Omega w\frac{\partial u}{\partial t}d\vec{x}=\int_\Omega wLud\vec{x}+\int_\Omega wfd\vec{x}.$$

Expanding our the L operator on u and splitting the integral over the sum, we find

$$\int_\Omega w\frac{\partial u}{\partial t}d\vec{x}=\sum_{\alpha,\beta=1}^{2}\int_\Omega wA^{\alpha\beta}\frac{\partial}{\partial x^\alpha}\left[\frac{\partial u}{\partial x^\beta}\right]d\vec{x}+\sum_{\gamma=1}^{2}\int_\Omega wB^\gamma\frac{\partial u}{\partial x^\gamma}d\vec{x}+\int_\Omega wCud\vec{x}+\int_\Omega wfd\vec{x}.$$

We notice that w is always independent of time so that the first integral can be rewritten as

$$\int_\Omega w\frac{\partial u}{\partial t}d\vec{x}=\frac{\partial}{\partial t}\int_\Omega wud\vec{x}.$$

We can now apply integration by parts on the second integral, and rewrite this as

$$\int_\Omega wA^{\alpha\beta}\frac{\partial}{\partial x^\alpha}\left[\frac{\partial u}{\partial x^\beta}\right]d\vec{x}=\int_{\partial\Omega}A^{\alpha\beta}w\frac{\partial u}{\partial x^\beta}d\vec{x}-\int_\Omega\frac{\partial A^{\alpha\beta}w}{\partial x^\alpha}\frac{\partial u}{\partial x^\beta}d\vec{x}.$$

Since our restriction on the selection of of test functions w, we know that $w(\vec{x})=0\ \forall\vec{x}\in\partial\Omega$. Thus the integral over the boundary must be zero. Using this simiplification, we obtain the representation of the continuous variational formulation of the problem to be

(4.1)

$$\frac{\partial}{\partial t}\int_\Omega wud\vec{x}=-\sum_{\alpha,\beta=1}^{2}\int_\Omega\frac{\partial A^{\alpha\beta}w}{\partial x^\alpha}\frac{\partial u}{\partial x^\beta}d\vec{x}+\sum_{\gamma=1}^{2}\int_\Omega B^\gamma w\frac{\partial u}{\partial x_\gamma}d\vec{x}+\int_\Omega Cwud\vec{x}+\int_\Omega wfd\vec{x}.$$

5. Discretization

Now that we have a variational formulation of the partial differential equation, we wish to discretize the equation, so that it is possible to apply the method of finite elements.

In order to discretize the problem, we must have an understanding of the space that the solution resides in. It is clear that the solution to the equation will be in the Sobolev space $W^{1,2}(\Omega)$. The Sobolev space is the set of functions functions that are infinitely differentiable. In this case we can tell that our solution resides in the specific Sobolev space $W^{1,2}(\Omega)$. It is also important to mention that this can be written as $H^1(\Omega)$, because this Sobolev space is also a Hilbert space. A Hilbert space is one that possess the structure for inner product. Thus we are able to conclude that our solution exists in a space that supports inner product, and is infinitely differentiable.

Since the process of finite element methods is inherently intended for computation, problems arise when infinities come into play. Since the span of the Sobolev space $H^1(\Omega)$ is

infinite, we are required to consider a subspace of it. This will be the subspace of the Sobolev space that our approximation of the solution lives in, we will call this subspace $\widetilde{V} \subset H^1(\Omega)$.

We will construct a finite set of basis functions for the subspace $\widetilde{V}$ to be $\{\varphi_j\}$. For now we will leave these basis functions as arbitrary functions, but the process for the construction of the basis functions will be presented in a later section.

The approximation of u we define as u, since the approximation should converge to the exact solution, utilizing the same notation does not cause any issues. We construct this approximation such that it is an element of the subspace of the Sobolev space as we have defined. That is to say

$$u \in \widetilde{V} \subset H^1(\Omega)$$

Since our approximation is constructed in $\widetilde{V}$, then it must be expressible as a linear combination of the basis function φ_j of the finite subspace $\widetilde{V}$ that $\widetilde{u}$ is an element of. Thus we can express $\widetilde{u}$ as

$$u = \sum_{j=1}^{N} u_j(t)\varphi_j(\vec{x}), \tag{5.1}$$

where $u_j(t)$ is a function of time that once it is found, it is possible to construct the final approximation. Using this expression for our approximation the goal of the method is to determine the functions $u_j(t)$.

We can now substitute our expression for u into the continuous variational formulation of the problem (4.1).

(5.2)

$$\begin{aligned}\frac{\partial}{\partial t}\int_\Omega w\left[\sum_{j=1}^{N} u_j\varphi_j\right]d\vec{x} = &-\sum_{\alpha,\beta=1}^{2}\int_\Omega \frac{\partial A^{\alpha\beta}w}{\partial x^\alpha}\frac{\partial}{\partial x^\beta}\left[\sum_{j=1}^{N} u_j\varphi_j\right]d\vec{x}\\ &+\sum_{\gamma=1}^{2}\int_\Omega B^\gamma w\frac{\partial}{\partial x^\gamma}\left[\sum_{j=1}^{N} u_j\varphi_j\right]d\vec{x}+\int_\Omega Cw\left[\sum_{j=1}^{N} u_j\pi_j\right]d\vec{x}+\int_\Omega wf d\vec{x}.\end{aligned}$$

It is possible to rewrite this expression, with several simplifications. First is to note that since w and φ_j are independent of t, and u is independent of $\vec{x}$. Using this it is possible to rewrite the first term, by moving constants with respect to the integral (u_j) out of the integral, and then to move constants with respect to the derivative (the integral) out of the derivative. Thus we can write the first term as

$$\frac{\partial}{\partial t}\int_\Omega w\left[\sum_{j=1}^{N} u_j\varphi_j\right]dx\vec{x} = \sum_{j=1}^{N}\left[\frac{\partial u_j}{\partial t}\int_\Omega w\varphi_j d\vec{x}\right].$$

Similar process can be applied to the other terms in equation 5.2. By pulling the constants out of the integrals, and splitting the integral over the summation, we are able to rewrite

the expression as

(5.3)
$$\sum_{j=1}^{N}\left[\frac{\partial u_j}{\partial t}\int_\Omega w\varphi_j d\vec{x}\right] = -\sum_{j=1}^{N}\left[u_j\sum_{\alpha,\beta=1}^{2}\int_\Omega \frac{\partial A^{\alpha\beta}w}{\partial x^\alpha}\frac{\partial \varphi_j}{\partial x^\beta}d\vec{x}\right] + \sum_{j=1}^{N}\left[u_j\sum_{\gamma=1}^{2}\int_\Omega B^\gamma w\frac{\partial\varphi_j}{\partial x^\gamma}d\vec{x}\right] + \sum_{j=1}^{N}\left[u_j\int_\Omega Cw\varphi_j d\vec{x}\right] + \int_\Omega wf d\vec{x}.$$

Now, since we have not placed any restrictions on w, and it can be any function, we will chose the basis functions φ_i as our test functions. Substituting that into the expression we obtain the discretized version of the variational formulation of the problem.

(5.4)
$$\sum_{j=1}^{N}\left[\frac{\partial u_j}{\partial t}\int_\Omega \varphi_i\varphi_j d\vec{x}\right] = -\sum_{j=1}^{N}\left[u_j\sum_{\alpha,\beta=1}^{2}\int_\Omega \frac{\partial A^{\alpha\beta}\varphi_i}{\partial x^\alpha}\frac{\partial \varphi_j}{\partial x^\beta}d\vec{x}\right] + \sum_{j=1}^{N}\left[u_j\sum_{\gamma=1}^{2}\int_\Omega B^\gamma\varphi_i\frac{\partial\varphi_j}{\partial x^\gamma}d\vec{x}\right] + \sum_{j=1}^{N}\left[u_j\int_\Omega C\varphi_i\varphi_j d\vec{x}\right] + \int_\Omega \varphi_i f d\vec{x}.$$

It can be useful to examine this equation using the inner products as they act upon functions. That is to say, that we use the L^2 inner product defined as

$$\langle f|g\rangle_{L^2} = \int_\Omega fg d\vec{x}.$$

Using this notation, the expression can be rewritten to be

$$\sum_{j=1}^{N}\langle\varphi_i|\varphi_j\rangle\frac{\partial u_j}{\partial t} = -\sum_{j=1}^{N}\sum_{\alpha,\beta=1}^{2}\left\langle\frac{\partial}{\partial x^\alpha}\left[A^{\alpha\beta}\varphi_i\right]\middle|\frac{\partial\varphi_j}{\partial x^\beta}\right\rangle u_j + \sum_{j=1}^{N}\sum_{\gamma=1}^{2}\left\langle B^\gamma\varphi_i\middle|\frac{\partial\varphi_j}{\partial x^\gamma}\right\rangle u_j + \sum_{j=1}^{N}\langle C\varphi_i|\varphi_j\rangle u_j + \langle\varphi_i|f\rangle.$$

It is now possible to factor out the u_j from the right hand side summations. This will provide the equation

$$\sum_{j=1}^{N}\langle\varphi_i|\varphi_j\rangle\frac{\partial u_j}{\partial t} = \left[-\sum_{j=1}^{N}\sum_{\alpha,\beta=1}^{2}\left\langle\frac{\partial}{\partial x^\alpha}\left[A^{\alpha\beta}\varphi_i\right]\middle|\frac{\partial\varphi_j}{\partial x^\beta}\right\rangle + \sum_{j=1}^{N}\sum_{\gamma=1}^{2}\left\langle B^\gamma\varphi_i\middle|\frac{\partial\varphi_j}{\partial x^\gamma}\right\rangle + \sum_{j=1}^{N}\langle C\varphi_i|\varphi_j\rangle\right]u_j + \langle\varphi_i|f\rangle.$$

Since it is possible to replace w with *any* of the basis functions, that means that $i = 1, 2, \ldots, N$. This provides us with a system of equations, which is then expressible in terms of matrices. The matrix representation of the system is

(5.5)
$$G\partial_t U = \left(\overline{A} + \overline{B} + \overline{C}\right)U + F.$$

Where U is a vector in $\mathbb{R}^N$ composed of the u_j functions. We use the notation ∂_t to denote the operator of taking the partial derivative of the U vector with respect to time, element wise. The elements of G, $\overline{A}$, $\overline{B}$, $\overline{C}$, and F are defined as

$$\begin{aligned}
G_{ij} &= \langle \varphi_i | \varphi_j \rangle && i,j = 1,\dots,N \\
\overline{A}^{ij} &= -\sum_{\alpha,\beta=1}^{2} \left\langle \frac{\partial}{\partial x^\alpha}\left[A^{\alpha\beta}\varphi_i\right] \middle| \frac{\partial \varphi_j}{\partial x^\beta} \right\rangle && i,j = 1,\dots,N \\
\overline{B}^{ij} &= \sum_{\gamma=1}^{2} \left\langle B^\gamma \varphi_i \middle| \frac{\partial \varphi_j}{\partial x^\gamma} \right\rangle && i,j = 1,\dots,N \\
\overline{C}^{ij} &= \langle C\varphi_i | \varphi_j \rangle && i,j = 1,\dots,N \\
F &= \langle \varphi_i | f \rangle && i,j = 1,\dots,N.
\end{aligned}$$

It is often simpler to consider a matrix M such that $M = \overline{A} + \overline{B} + \overline{C}$, then our expression for the matrix form of the system of equations becomes

$$G\partial_t U = MU + F. \tag{5.6}$$

This matrix representation is how we will continue to consider the problem. For most cases this is the system of equations that will be found. However, this expression is only sufficient for Dirichlet boundary conditions, as the integral over the boundary for other boundary conditions would not be zero. Thus there would be additional terms in the system of equations. This is not an issue with the Dirichlet boundary conditions, so we do not need to consider any additional terms in the matrix representation.

If the problem that is being considered were to be time independent, then the matrix formulation of the system of equations would be

$$MU = F.$$

We will continue to consider both the time dependent and the time independent problems, as both are useful in different situations. However, they are solved in different ways, some simplifications can be made with one but not the other.

6. Localization

While is is ocassionaly possible to find the elements of the matricies in equation 5.6 on the global domain, this proves to be exceedingly difficult in most if not all cases. To remediate this issue, we implement a generalizable method that will work for any domain and and functions that are being considered. Since we have the abbility to select any arbitrary baseis functions as per our definition of the approximation to the solution, then it is possible to selece a basis wuch that it it becomse possible to split the domain into a finite number of subdomains.

We first discretize the domain Ω into N mutually exclusive subdomains. Each of these subdomains is called an *element*. For the purposes of this paper, each element will be defined to be a triangle subdomain of Ω. The set of all elements is called the triangular decomposition or the mesh of the domain, the process for constructing the triangular decomposition is discussed further in section 7. For the localization process, the specifics of the triangular decomposition is not necessary, so we will not go in depth on the structure of the triangular decomposition in this section.

For each element of the mesh, we denote the element as $E^{(e)}$, where $e = 1, \ldots, N$. Then each vertex of the triangle is labeled as $E_i^{(e)}$ where $i = 1, 2, 3$. The notation for the x and y values of each vertex we denote as $X_i^{(e)}$ and $Y_i^{(e)}$ respectively, where $i = 1, 2, 3$.

In 1D the process is trivial as all element have the same dimensions, but in higher dimensional implementation, each element can have variable shapes and sizes. This becomes more difficult because integrating over each element would require entirely different process each time to account for the differences in shape and size. To resolve this issue, we construct a master space, which is possible to transition to from any element, and back to each element. Thus we only need determine a method to integrate over our master element.

6.1. **Master Space.** The master space is a domain that is constructed in order to simplify the construction of the local approximations. For the situation with triangular elements, we construct the master space such that the vertices of a triangle lie on the points $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$. Then for each element in the mesh, we construct a transformation between the master space and the element space.

We define the master element to be $\widehat{E}$. An image of the master element can be seen in figure **??**.

The master element utilizes what is called a Barycentric coordinate system. The barycentric coordinates are demonstrated by each vertex being its own dimension, so if the element were to be quadrangles, then the barycentric coordinates would be in $\mathbb{R}^4$ instead of $\mathbb{R}^3$ as they are for triangles. The conversion between the global coordinates, and the Barycentric coordinates can be done using the following equations

$$\begin{aligned}
\lambda_1 &= \frac{(Y_2 - Y_3)(x - X_3) + (X_3 - X_2)(y - Y_3)}{(Y_2 - Y_3)(X_1 - X_3) + (X_3 - X_2)(Y_1 - Y_3)} \\
\lambda_2 &= \frac{(Y_3 - Y_1)(x - X_3) + (X_1 - X_3)(y - Y_3)}{(Y_2 - Y_3)(X_1 - X_3) + (X_3 - X_2)(Y_1 - Y_3)} \\
\lambda_3 &= 1 - \lambda_1 - \lambda_2.
\end{aligned}$$

Where X_i and Y_i represent the x and y coordinates of the ith vertex of the element. x, y are the coordinates of the point in the triangle that is to be converted to the Barycentric coordinates. Converting from Barycentric coordinates to the global coordinates uses the following equations

$$\begin{aligned}
x &= \lambda_1 X_1 + \lambda_2 X_2 + \lambda_3 X_3 \\
Y &= \lambda_1 Y_1 + \lambda_2 Y_2 + \lambda_3 Y_3
\end{aligned}$$

6.2. **Local Basis.** The first step is to construct a local basis function, that are only defined for the current element under consideration. There should be a basis element for every vertex in the element, so for triangular element, there will be three local basis functions. The notation for the local basis functions is $\varphi_1^{(e)}$, $\varphi_2^{(e)}$, and $\varphi_3^{(e)}$. To make computation simple, the construct the basis function such that they are linear, with the requirement that at the corresponding vertex, the local basis function is 1 and at all other vertices of the element the basis function will be 0.

The conversion from global space to the bilinear coordinates does exactly this task. Thus the definition of the local basis functions are

$$
\begin{aligned}
\varphi_1^{(e)} &= \frac{(Y_2^{(e)} - Y_3^{(e)})(x - X_3^{(e)}) + (X_3^{(e)} - X_2^{(e)})(y - Y_3^{(e)})}{(Y_2^{(e)} - Y_3^{(e)})(X_1^{(e)} - X_3^{(e)}) + (X_3^{(e)} - X_2^{(e)})(Y_1^{(e)} - Y_3^{(e)})} \\
\varphi_2^{(e)} &= \frac{(Y_3^{(e)} - Y_1^{(e)})(x - X_3^{(e)}) + (X_1^{(e)} - X_3^{(e)})(y - Y_3^{(e)})}{(Y_2^{(e)} - Y_3^{(e)})(X_1^{(e)} - X_3^{(e)}) + (X_3^{(e)} - X_2^{(e)})(Y_1^{(e)} - Y_3^{(e)})} \\
\varphi_3^{(e)} &= 1 - \varphi_1^{(e)} - \varphi_2^{(e)}.
\end{aligned} \tag{6.1}
$$

Thus for every element there exists three local basis functions. A plot of these local basis functions is shown in figure **??** on an arbitrary triangular element.

6.3. **Global Basis.** The next step is to construct the global basis functions. Similarly to the strategy for the construction of the local basis functions, each global basis will be associated with a single vertex of the mesh. Thus for a mesh with N elements, then there will be some number of global basis functions less than $3N$. The method to define the global basis function is to define it to be 1 at the associated vertex, and 0 at all other vertices in the mesh.

In order to achieve this, the global basis function can be constructed as a piecewise combination of k different local basis functions, where k is the number of element that share a given vertex. The exact formulation of the global basis functions is dependent on the triangular mesh, and the current vertex index. An example of what a global basis function could be like is depicted in figure **??**. One can intuitively think of the global basis functions as k sided pyramids this their peak directly above the associated vertex.

6.4. **Local System.** Using the definition of local basis functions that were constructed in section 6.2, the construction of an element-specific system of equations is preformed. The system of equations is identical to the global system, but is simply specified to a single element. Thus the local system of equations in matrix form is

$$
G^{(e)} \partial_t U^{(e)} = \left(\overline{A}^{(e)} + \overline{B}^{(e)} + \overline{C}^{(e)} \right) U^{(e)} + F^{(e)}.
$$

In this system of equations, the elements of each matrix are only determined by the local basis function on the element e. The size of the matrix is equivalent to the number of vertices in the element, thus for the triangular element case, the matrices have a size of three. The

element of these matrices can be found through the following equations

$$
\begin{aligned}
G_{ij}^{(e)} &= \int_{E_e} \varphi_i^{(e)} \varphi_j^{(e)} d\vec{x} && i,j = 1,2,3 \\
\overline{A}_{ij}^{(e)} &= -\sum_{\alpha,\beta=1}^{2} \int_{E_e} \frac{\partial \varphi_j^{(e)}}{\partial x^\beta} \frac{\partial}{\partial x^\alpha} \left(A_{\alpha\beta} \varphi_i^{(e)} \right) d\vec{x} && i,j = 1,2,3 \\
\overline{B}_{ij}^{(e)} &= \sum_{\gamma=1}^{2} \int_{E_e} B^\gamma \varphi_i^{(e)} \frac{\partial \varphi_j^{(e)}}{\partial x^\gamma} d\vec{x} && i,j = 1,2,3 \\
\overline{C}_{ij}^{(e)} &= \int_{E_e} C \varphi_i^{(e)} \varphi_j^{(e)} d\vec{x} && i,j = 1,2,3 \\
F_i^{(e)} &= \int_{E_e} \varphi_i^{(e)} f d\vec{x} && i = 1,2,3.
\end{aligned}
\tag{6.2}
$$

Using only the local matrix elements, it is possible to construct the global system, utilizing some specific algorithms that are discussed in section 8.

7. Mesh Generation

For the implementation of finite element methods it is required to first discretize the global domain into a set of finite subdomains. Each subdomain is called an element. It is then possible to compute the values of the local system of equations with respect to a given element, then to combine these values to construct the global system. The first step is to discretize the global domain into a set of finite elements.

These elements can be of the form of an N sided polygon, not that it will not be regular each side will have different lengths. This paper will utilize triangular meshes, but many implementations also implement quadrangle, and hexagonal meshes. Since the construction of meshes with elements with more than three sides is considerably more difficult, we will limit this paper to only discussing the triangular mesh construction.

Constructing the triangular mesh is the most generalized method, as any polygon with more edges can be constructed by several triangles. Thus the mesh of polygons with more than three sides, can be derived from the triangular mesh, without significant difficulty.

The process that will be used for the construction of the triangular mesh is knows as Constrained Delaunay Triangulation. Then to improve the mesh and consequently improve the accuracy of the approximation, a mesh refinement algorithm is utilized to ensure that the elements of the mesh are of an acceptable quality.

7.1. Delaunay Triangulation.

Delaunay triangulation is the straight line dual of the Voronoi diagram. The Delaunay triangulation of a domain is commonly used in may different situations, including for the purpose of mesh generation for finite element analysis.

The key property of Delaunay triangulation is that no point may lie within the circumcircle of any other triangle.

Definition 7.1 (Circumcircle)**.** The circumcircle of a triangle, is the circle is defined by having all three vertices of the triangle on the circle. For any given triangle there is only one such circle that satisfies this. In figure **??**, a depiction of the circumcircle for a provided triangle is given.

Definition 7.2 (Delaunay Triangulation)**.** Let S be a set of points in the plane $\mathbb{R}^2$. A triangulation T is a *Delaunay Triangulation*(DT) of S if for each triangle t of T there exists a circle C with the following properties:

(1) all of the vertices of the triangle t are on the boundary of circle C,

$$p \in \partial C \quad \forall p \in t$$

(2) and no other vertex of S is in the interior of C.

$$p \notin S \quad \forall p \in \mathrm{Int}(C)$$

By this definition of the Delaunay triangulation, the triangulation is unique for most set of points S. The only exception to this rule is when a set of four points construct the square. For the square there are two equally valid triangulations, and dependent on the implementation one or the other must be selected.

A major advantage of the Delaunay triangulation for mesh generation is that it inherently avoids triangles with small included angles, as the circumcircles of these triangles would be very large, and would likely include other vertices. This avoidance is extremely useful in finite element analysis, and will produce more accurate approximations of the solution to the partial differential equation.

There are many different algorithms that can be implemented for the construction of the Delaunay triangulation. The notable methods include; Divide and Conquer, Incremental, Sweep Line, and Edge Flipping. The specifics of these algorithms will be discussed further in section **??**.

With the Delaunay triangulation constructed, one must next apply the boundaries of the domain to the mesh. The process is called the construction of the Constrained Delaunay Triangulation.

7.2. **Constrained Delaunay Triangulation.** The Constrained Delaunay Triangulation, is a modification of the Delaunay Triangulation such that specified edges and vertices are forced to exists in the triangulation. Because of the forcing of edges into the triangulation, the mesh may not strictly satisfy the definition of Delaunay triangulation. Thus we define the constrained triangulation as the closest triangulation to the Delaunay triangulation that includes all of the required edges.

This sense of "closeness" to the Delaunay mesh is such that we use as few modifications from a Delaunay triangulation in order to enforce the constrained edges to construct the constrained Delaunay triangulation.

The constrained edges are most commonly utilized for the purpose of defining boundaries of the domain, or enforcing a location of a medium change. Since the result of the unconstrained Delaunay triangulation will have a strictly convex shape. This makes it necessary to apply the edge constraints for all domains that do not have a strictly convex outer hull.

There are a number of methods for the construction of the constrained Delaunay triangulation. A few of these implementations enforce the constraints during the initial construction of the triangulation. However, more commonly the constraints are enforced upon an already constructed triangulation. The details on the implementation of these algorithms is discussed in section **??**.

7.3. **Mesh Refinement.** The direct output of the constrained Delaunay triangulation algorithms may be sufficient in a few situations, but for most cases, it would result in suboptimal

meshes. For generalizability, we discuss the methods that are utilized to refine the generated mesh, thus improving upon the quality of the meshes.

Mesh refinement is almost always required, because along constrained edges, there may only be a single element. This element is forced to have one of its edges be the entity of the constrained edge. This tends to lead to skinny triangles, ones who's minimum interior angle is very small. There are frequently undesirable in the final mesh. We denote that these triangles are not *well-shaped*.

Definition 7.3 (Well-Shaped)**.** A triangle is well-shaped if all its interior angles are greater than or equal to some defined constant angle α (commonly $\alpha \approx 30°$).

It is also possible that all the elements in the mesh are well-shaped, but they could all be very large. A finer mesh will always result in more accurate approximations, as will be made clear later. Thus we may also want to define a notion of *well-sized* triangles, so that it is possible to refine the element size in the mesh.

Definition 7.4 (Well-Sized)**.** A triangle is well-sized if the area of the triangle satisfies some user defined grading function g (commonly $g = \text{const}$). This function can use any criteria for determining if a triangle satisfies it, as long as there exists a value $\delta > 0$ such that any well-shaped triangle that fits within a circle of radius δ would satisfy the grading function.

This definition of the well-sized triangles, describes that any function can be used in order to define when a triangle is well-sized. This is very important in constructing meshes that are very fine around specified features, and the coarser in the unimportant regions. The one restriction of the grading function is that there must be some lower bound, such that all triangles that fit within the lower bound will be accepted. This enforces that the grading function may not require triangles to become infinitely small.

Using these definitions for how to classify the elements of a mesh, it becomes possible to implement algorithms in order to refine any mesh. The technicalities of the mesh refinement algorithms is discussed in **??**.

8. Assembly Of The Global System

Although it is possible to directly compute the elements of the global matrices, it is ineffective and not practical for any non trivial problems. Thus we will not consider the explanation of how to construct the global matrices directly.

The method that we utilize is an incremental construction of the global matrix, through the addition of elements from each of the local matrices. We begin by initializing the matrix G, $M = \overline{A} + \overline{B} + \overline{C}$ and F of zeros. G and M are $N \times N$ matrices, and F is a $N \times 1$ matrix, where N is the number of vertices in the triangulation.

Every element matrix $G^{(e)}$ and $M^{(e)}$ are 3×3 matrices, and each element matrix $F^{(e)}$ is a 3×1 matrix. We need to construct a method that will allow us to associate the elements of the local matrix to the elements of the global matrix. The first part of this is determining a method to associate the element vertex to the global vertex. We define a function *node* to do just this

$$node : \mathbb{R} \times \{1, 2, 3\} \to \mathbb{R}$$
$$node(e, I) = i.$$

This construction of the *node* function, is such that given a element index, and the local index of a vertex in the specified element, it will return the global vertex number.

Using this function, it is possible to construct the algorithm to assemble the global system of equations. The general method is to add each local matrix element to the associated global matrix element, where the association is done using the *node* function. This can be viewed as

$$A_{node(e,I)node(e,J)} = A_{node(e,I)node(e,J)} + A_{IJ}^{(e)} \quad I, J = 1, 2, 3.$$

This is done for all of the elements in the mesh, and all of the local matrix elements for each mesh element.

More specifics of the algorithm for the global matrix construction are discussed in section **??**.

For the time independent problem statements, this is the extent of what needs to be done for the assembly of the global system of equations. It provides us with an expresion of the form

$$MU = F,$$

which can easily be solved for using any number of algorithms that will be discussed in section **??**. However, for the time dependent problems, more must be done to prepare for time iterations.

For the time dependent expressions, we need to be able to solve the system of equations imperatively. For this one can use any number of methods, such as explicit Euler, implicit Euler, Runge-Kutta, or Crank-Nicolson. For now we will use the Crank-Nicolson method of iteration, as it provides significantly more accurate results than either Euler method, and is significantly more simplistic compared to Runge-Kutta.

8.1. **Euler.** For the time dependent iteration, it is important to note that both U and F will be dependent on the time, or in this case the current iteration. These are the only matrices that are dependent on the time, because in our expressions for the evaluation of the elements of the matrices (equation 6.2), the only elements that have a time dependent function is that of F, thus F is the only input matrix that is dependent on time. And it is clear that if F is dependent on the iteration, that U will also be dependent on the iteration.

Because of this dependence on the iteration, we will denote the current iteration by using a superscript n. Thus we can rewrite U and F as U^n and F^n respectively.

The basis of the time iteration, is to use the approximation

$$\frac{\partial U}{\partial t} \approx \frac{U^{n+1} - U^n}{\Delta t}.$$

This approximation is directly derived from the definition of a derivative, and it is clear that as $\Delta t \to 0$ then the approximation becomes an equality.

Substituting this approximation into the matrix formulation of the system of equations, we find the new expression to become

$$G\frac{U^{n+1} - U^n}{\Delta t} = MU^n + F^n.$$

Now we solve this equation for U^{n+1} as it is safe to assume that U^n is already known from the previous iteration. Thus the expression

$$GU^{n+1} = (MU^n + F^n)\,\Delta t + GU^n$$
$$GU^{n+1} = (G + M\Delta t)\,U^n + F^n\Delta t$$

is found.

Using this expression, it becomes simple to compute the value of U^{n+1}. This is because, we can further rewrite this expression as

$$\begin{aligned} GU^{n+1} &= Q \\ \text{where } Q &= (G + M\Delta t)\, U^n + F^n \Delta t. \end{aligned}$$

It is well known how to solve a system of equations of the form $Ax = b$, so this form of the expression can be solve for the value of U^{n+1}.

Unfortunately the Euler method only provides first order accuracy, so we will instead utilize the Crank-Nicolson method of iteration.

8.2. **Crank-Nicolson.** For the Crank-Nicolson method of iteration, we follow a similar process to the Euler method of iteration. The main difference is that instead of using the old values of U and F on the right hand side of the expression, one uses the average of the old and the new values. This is written as

$$G\frac{U^{n+1} - U^n}{\Delta t} = M\frac{U^{n+1} + U^n}{2} + \frac{F^{n+1} + F^n}{2}.$$

Once again, we assume that U^n is known from the previous iterations, and F^{n+1}, and F^n can be computed as givens. Thus we solve for U^{n+1}. We find that the expression becomes

$$\left(G + \frac{1}{2}M\Delta t\right) U^{n+1} = \left(G - \frac{1}{2}M\Delta t\right) U^n + \frac{F^{n+1} + F^n}{2}\Delta t$$

In order to simplify this expression, we define three new matrices to be

$$\begin{aligned} \widetilde{A} &= G + \frac{1}{2}M\Delta t \\ \widetilde{B} &= G - \frac{1}{2}M\Delta t \\ \widetilde{C}^n &= \frac{F^{n+1} + F^n}{2}\Delta t. \end{aligned}$$

Now the expression be be rewritten as

$$\widetilde{A}U^{n+1} = \widetilde{B}U^n + \widetilde{C}^n.$$

Since U^n is assumed to be known from the previous iteration, we can define a new matrix Q^n to be

$$Q^n = \widetilde{B}U^n + \widetilde{C}^n.$$

Using this new matrix, we can rewrite the expression one final time to be

$$\widetilde{A}U^{n+1} = Q^n.$$

This is finally in the format that is well known how to solve for the values of U^{n+1}, so all that needs to be done is apply some algorithm to solve this linear system.

9. Dirichlet Boundary Conditions

Now that the global system of equations has been constructed, we are able to impose the predefined boundary conditions to the system of equations. As discussed previously, we will only consider the Dirichlet boundary conditions in this explanation.

The global system of equations can be written as

$$\widetilde{A}U^{n+1} = Q^n$$

for the time dependent problem, and as

$$MU = F$$

for the time dependent case. The format of each expression is very very similar, this instead of explicitly explaining the process of imposing the boundary conditions on the system for each, we will provide the process of imposing the boundary conditions for the time dependent case for generality, and the case for the time independent case will follow the exact same process.

We rewrite the global system of equations in the general form to be

$$\begin{pmatrix} \widetilde{A}_{11} & \widetilde{A}_{12} & \cdots & \widetilde{A}_{1N} \\ \widetilde{A}_{21} & \widetilde{A}_{22} & \cdots & \widetilde{A}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{A}_{N1} & \widetilde{A}_{N2} & \cdots & \widetilde{A}_{NN} \end{pmatrix} \begin{pmatrix} U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_N^{n+1} \end{pmatrix} = \begin{pmatrix} Q_1^n \\ Q_2^n \\ \vdots \\ Q_N^n \end{pmatrix}.$$

Currently this system of equations is not well behaved, as we have thus far neglected to prescribe the boundary conditions. This means that the matrix system of equations is overdetermined and cannot be solved. In order to fix this, we must apply the boundary conditions. Due to the Dirichlet boundary conditions, we know that for any vertex in the mesh that is on the boundary $x_i \in \partial\Omega$,

$$U_i^n = \partial u(x_i, n\Delta t)$$

Part 2. Computational

References

[BBC+93] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charels Romine, and Hen Van der Vorst. *Templates for the Solution of Linear Systems: Buildign Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, 2 edition, 1993.

[Che89] L. Paul Chew. Contrained delaunay triangulations. *Algorithmica*, 4:92–108, 1989.

[Che93] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. *9th Annual Symposium on Computation Geometry*, pages 275–280, 1993.

[Dv08] V. Domiter and B. Žalik. Sweep-line algorithm for constrained delaunay triangulation. *International Journal of Geographical Information Science*, 22(4):449–462, 2008.

[HKA12] Farzana Hussain, M. S. Karim, and Razwan Ahamad. Appropriate gaussian quadrature formulae for triangle. *International Journal of Applied Mathematics and Computation*, 4(1):24–38, 2012.

[KH15] Dmitri Kuzmin and Jari Hämäläinen. *Finite Element Methods for Computational Fluid Dynamics*. Society for Industrial and Applied Mathematics, Philadelphia, 2015.

[Kry31] A. N. Krylov. On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined. *News of Academy of Sciences of the USSR*, 7(4):491–539, 1931.

[Rup95] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[She96] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[She02] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21,74, 2002.

[Slo87] S. W. Sloan. A fast algorithm for construction delaunay triangulations in the plane. *Adv. Eng. Software*, 9(1):34–42, 1987.

[Slo93] S. W. Sloan. A fast algorithm for generating constrained delaunay triangulations. *Computers & Structures*, 47(3):441–450, 1993.

[SS86] Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *Sci. Stat. Comput.*, 7(3):856–869, 1986.

[Ž05] Borut Žalik. An efficient sweep-line delaunay triangulation algorithm. *Computer-Aided Design*, 37:1027–1038, 2005.