FINITE ELEMENT ANALYSIS

ARDEN RASMUSSEN

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.

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

$$\partial_t u = Lu + f,$$

where L is an operator of the form

$$L = A^{ij}\partial x_i \partial x_j + B^i \partial x_i + C$$

and f is some forcing function. We will also enforce that A^{ij} 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

$$\partial_t u = A^{ij}\partial x_i \partial x_j u + B^i \partial x_i + Cu + f \tag{1}$$

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 $u(\Gamma, t)$ is provided. Note that this could mean that the boundary is split into several

Date: April 2, 2019.

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^{ij}, B^i, 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.

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.