WEAK GALERKIN FINITE ELEMENT SOLVER WITH POLYGONAL MESHES

A Project Report Submitted

for the Course

MA499 Project II

by

Ansh Bhatt

(Roll No. 180123005) &

Naman Goyal

(Roll No. 180123029)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

April 2022

CERTIFICATE

This is to certify that the work contained in this project report entitled "Weak galerkin finite element solver using polygonal meshes" submitted by Ansh Bhatt (Roll No.: 180123005) and Naman Goyal (Roll No.: 180123029) to the Department of Mathematics, Indian Institute of Technology Guwahati towards partial requirement of Bachelor of Technology in Mathematics and Computing has been carried out by them under my supervision.

It is also certified that, along with literature survey, a few new results are established/computational implementations have been carried out/simulation studies have been carried out/empirical analysis has been done by the student under the project.

Guwahati - 781 039 (Dr. Bhupen Deka)

April 2022 Project Supervisor

ABSTRACT

In mathematics and various associated fields Finite Element Method is used to numerically approximate the solutions to partial differential equations which are not possible to solve analytically. In this project we try to develop an efficient solver for Finite Element approximations using polygonal meshes. We start with an introduction to the method, followed by its computational techniques and then proceed to look at the various aspects of an FEM solver.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Differential equations are used in a variety of sectors of science and engineering to represent mathematical models. The majority of the time, solving these differential equations analytically is extremely difficult. Numerical approaches that approximate the answer are required to obtain the solutions to these mathematical models. We are transitioning towards numerical methods that efficiently and accurately approximate the solutions to complex mathematical models as high-speed computers develop.

One of the numerical methods is the Finite Element Method. Instead of computing the solution of the model on the entire domain, FEM is utilised to do it in a finite number of sub-domains. The computational domain is broken down into sub-domains, each of which is called an element. The mesh is made up of all of these pieces combined together. At every location in the domain, FEM produces a numerical estimate of the model's solution.

After a general introduction to the method in which we look at the issue formulation and attempt to describe the mathematics behind the method, we will go on to using FEM to solve a boundary value problem. We'll learn

about the many aspects of creating a FEM solver in this section.

Let $\Omega \in \mathbb{R}^2$ be a bounded domain with smooth boundary $\partial\Omega$. We consider following elliptic boundary value problem

$$\left.\begin{aligned} -\Delta u &= f \text{ in } \Omega, \\ u &= 0 \text{ in } \partial\Omega. \end{aligned}\right\} \tag{1.1}$$

We need to study the convergence of finite element solution to the exact solution of (1.1) with respect to H^1-norm and L^2-norm.

1.1 Basic Notations

We shall now introduce the standard notation for Sobolev spaces and norms to be used.

Definition 1.1.1. Let $\Omega \in R^2$ and p is a real number with the property $1 \leq p \leq \infty$, *then* $L^p(\Omega)$ denotes the following space

$$L^p(\Omega) = \left\{ f : \Omega \to \mathbb{R} : \left(\int_\Omega |f(x)|^p dx \right)^{\frac{1}{p}} < \infty \right\}.$$

Further, $L^p\ (\Omega)$ is a normed linear space with respect to the following norm

$$\|f\|_{L^p(\Omega)} = \left(\int_\Omega |f(x)|^p dx \right)^{\frac{1}{p}}$$

Definition 1.1.2. Let m be a positive integer and $1 \leq p < \infty$. Then the Sobolev spaces $W^{m,p}(\Omega)$ is defined by

$$W^{m,p} = \{u \in L^p(\Omega) : D^\alpha u \in L^p(\Omega), 0 \leq \alpha \leq m\},$$

where
$$D^\alpha u = \frac{\partial^\alpha u}{\partial^{\alpha_1} x_1 \partial^{\alpha_2} x_2},\ \alpha = \alpha_1 + \alpha_2$$

is the weak derivative of u at the αth order. To put it another way, $W^{m,p}(\Omega)$ is a collection of all functions in $L^p\ (\Omega)$ that have all of their weak derivatives up to order m in $L^p\ (\Omega)$. In addition, $W^{m,p}(\Omega)$ is a normed linear space in terms of the norm.

$$\|v\|_{m,p,\Omega} = \sum_{0\leq\alpha\leq m} \|D^\alpha v\|_{L^2(\Omega)}$$

For our later use, we introduce following semi-norm

$$|v|_{m,p,\Omega} = \sum_{\alpha=m} \|D^\alpha v\|_{L^2(\Omega)}$$

Now in order to introduce the weak derivative, we consider the following equation

$$\frac{dy}{dx} = g(x)\ in\ \Omega \in \mathbb{R}$$

It is well understood that if $g \in C(\Omega)$, then $y \in C^1(\Omega)$. If $g \in L^2(\Omega)$, the difficulty emerges, and we cannot expect a solution in $C^1(\Omega)$. Of course, we suppose that $y \in C(\Omega) \supset C^1(\Omega)$. C($\Omega$) functions, on the other hand, do not have to be differentiable. As a result, we can't find $\frac{dy}{dx}$ in the traditional sense. As a result, even if y is neither differentiable or at all continuous, we must give the derivative of y a meaning. The weak derivative of such functions is used to do this.

Let us try to define the weak derivative of a differentiable function *f*, and then generalise that definition to a bigger class of functions to encourage us to look for weak derivatives of such functions. Let $C_0^\infty(\Omega)$ be the collection of all C^∞ functions defined over Ω that vanish at the boundary $\partial\Omega$ of Ω. Then we obtain

$$\int_\Omega \frac{df}{dx} v dx = fv|_{\partial\Omega} - \int_\Omega f\frac{dv}{dx}dx = -\int_\Omega f\frac{dv}{dx}dx$$

for $v \in C_0^\infty(\Omega)$. As a result, under the sign of integration, the derivative of f decreases to the derivative of v. We define non-smooth functions' weak derivatives in this way. Before we go into the formal definition of weak derivatives, let's speak about function support.

Definition 1.1.3. Let ϕ be a real(or complex) valued continuous functions on an open set $\Omega \in \mathbb{R}^n$. The support of ϕ is denoted by supp(ϕ) and defined as

$$supp(\phi) = \overline{\{x \in \Omega : \phi(x) \neq 0\}}$$

If this closed set is compact as well, then supp(ϕ) is said to be of compact support.

The set of all infinitely differentiable function on an open set $\Omega \in \mathbb{R}^n$ with compact support is a vector space which will henceforth be denoted by $D(\Omega)$. Therefore, first order weak derivative of f is given by

$$D_f : D(\Omega) \to \mathbb{R} \; such \; that \; D_f(v) = -\int_\Omega f v' dx \; \forall v \in D(\Omega)$$

We can construct the weak derivative for those functions f that are neither differentiable or continuous because the right hand side does not require the derivative of *f*. The *mth* order weak derivative of f is generally defined as

$$D_f^m(v) = (-1)^m \int_\Omega \frac{d^m v}{dx^m} f dx \; \forall v \in D(\Omega)$$

Definition 1.1.4. For p = 2, the Sobolev space $W^{m,p}(\Omega)$ is a Hilbert space and it is denoted by $H^m(\Omega)$. In particular, $H^1(\Omega) = \{v \in L^2(\Omega) : Dv \in L^2(\Omega)\}$ and the corresponding norm is defined by

$$\|v\|_{H^1(\Omega)} = \|v\|_{L^2(\Omega)} + \|Dv\|_{L^2(\Omega)}$$

Definition 1.1.5. It can be proved that $C_0^\infty(\bar{\Omega})$ is dense in $H^1(\Omega)$. If $f \in C^\infty(\bar{\Omega})$ we define the trace of f, namely γf, by $\gamma f = f|_\Gamma, \Gamma = \partial\Omega$. Further the map $\gamma : C^\infty(\bar{\Omega}) \to L^2(\Gamma)$ is continuous and linear satisfying

$$\|\gamma u\|_{L^2(\Gamma)} \leq C\|u\|_{H^1(\Omega)}$$

Hence this can be extended as continuous linear map from $H^1(\Omega)$ to $L^2(\Gamma)$. This map γ is called trace map.

Definition 1.1.6. The collection of all H^1 functions vanishing on the boundary is a closed subspace of H^1 and it is denoted by

$$H_0^1(\Omega) = \{v \in H^1(\Omega) : \gamma v = 0\}$$

. For $H_0^1(\Omega)$ functions, following important inequality, due to Poincar´e, holds true

$$\int_\Omega |v|^2 dx \leq C \int_\Omega |Dv|^2 dx \ \forall v \in H_0^1(\Omega)$$

We'll wrap up this part by talking about Sobolev quotient space. Let $P_k(\Omega)$ be the set of all polynomials of degree less than or equal to k ≥ 0, defined over Ω . We define

$$V = W^{k+1,p}(\Omega)/P_k(\Omega)$$
$$= \{[v]|[v] = \{v + q|q \in P_k(\Omega)\}, v \in W^{k+1,p}(\Omega)\}$$

and its corresponding norm

$$\|[v]\|_V = \inf_{p \in P_k(\Omega)} \|v + q\|_{k+1,p,\Omega}$$

Lemma 1.1.7. *For any Lipschitz domain $\Omega \subset \mathbb{R}^d$, there is a constant $C > 0$, depending only on Ω, such that*

$$\inf_{p \in P_k(\Omega)} \|v + p\|_{k+1,\Omega} \leq c|v|_{k+1,\Omega} \ \forall v \in H^{k+1}(\Omega) \tag{1.2}$$

1.2 Variation Formulation of Elliptic BVP

We present a weak formulation for an elliptic boundary value problem in this section. Furthermore, the Lax-Milgram finding is used to investigate the variational problem's well-posedness.

Multiply (1.1) by a smooth function $v \in D(\Omega)$, where $D(\Omega)$ is the set of all infinitely differentiable function on Ω with compact support, and then integrate over Ω to have

$$\int_\Omega -\triangle u.vdx \ = \int fvdx \tag{1.3}$$

Now, we may recall classical Green's theorem

Theorem 1.2.1. ***Green's Theorem.*** *Let Ω be a connected domain in R^2. Then, we have*

$$-\int_\Omega \nabla.(\nabla G)wdxdy \ = \int_\Omega \nabla G.\nabla wdxdy - \oint_\Gamma \frac{\partial G}{\partial \eta}$$

where η is the outward normal to the boundary $\partial\Omega = \Gamma$.

Now, apply Green's theorem and the fact v = 0 on $\partial\Omega$ in (1.3) to have

$$\int_\Omega \nabla u.\nabla vdx \ = \int_\Omega fvdx$$

Then variational problem to (1.1) is defined as: Find $u \in H_0^1(\Omega)$ such that

$$A(u, v) = L(v) \ \forall v \in H_0^1(\Omega) \tag{1.4}$$

Here, A : $H_0^1(\Omega) \times H_0^1(\Omega) \rightarrow$ R and L : $H_0^1(\Omega) \rightarrow \mathbb{R}$ are defined as

$$A(u,v) = \int_\Omega \nabla u.\nabla v dx \ \forall u,\ v \in H_0^1(\Omega)$$

$$L(v) = \int_\Omega f v dx \ \forall v \in H_0^1(\Omega)$$

The variational issue is approximated using the finite element approach (1.4). As a result, the existence and uniqueness of the variational problem become extremely crucial. In this regard, we must comprehend the Lax-Milgram Lemma, which is a well-known finding. Before we go any further, let's define a few key words.

Definition 1.2.2. Billinear Form: If X and Y are vector spaces, a billinear form A : X × Y → R is defined to be an operator with following properties

$$A(\alpha u + \beta w, v) = \alpha A(u,v) + \beta A(w,v) \ \forall u, w \in X, v \in Y, \alpha, \beta \in \mathbb{R}$$

$$A(u, \alpha v + \beta w) = \alpha A(u,v) + \beta A(u,w) \ \forall u \in X, v, w \in Y$$

Definition 1.2.3. Continuous Billinear Map: Let A : X×Y → ($\mathbb{R}$) be a billinear map, where X and Y are normed linear spaces equipped with norms $\|.\|_X$ and $\|.\|_Y$, respectively. Then A(., .) is said to be continuous/bounded if there is a positive number C such that

$$|A(u,v)| \leq C\|u\|_X \|v\|_Y \ \forall u \in X, v \in Y$$

Definition 1.2.4. H-Elliptic Billinear Map/ Positive Billinear Map:

Given a billinear form A : H × H$\mathbb{R}$, where H is an inner product space, we say that A(., .) is H-elliptic/positive if there exist a constant C > 0 such that

$$A(v,v) \geq C\|v\|_H^2 \ \forall v \in H$$

where $\|.\|$ is the norm associated with inner product.

We can now move to discussing the key finding in this chapter, which offers conditions that are necessary for the existence and uniqueness of the variational problem, now that we have presented all of the terminology.

Theorem 1.2.5. *Let H be a Hilbert space and let $A : H \times H \to \mathbb{R}$ be a continuous, positive billinear map defined on $H \times H$. Then, for a given continuous linear functional L on H, there exist a unique element u in H such that*

$$A(u, v) = L(v) \ \forall v \in H$$

1.3 Regularity Result

Depending on the smoothness of the available data, this section describes the existence, uniqueness, and smoothness of the issue (1.4) solutions. Consider the following equation for clarity.

$$\frac{dy}{dx} = f, x \in (a, b)$$

If $f \in L^2(a, b)$ then $\frac{dy}{dx} \in L^2(a, b)$ so y $\in H^1(a, b)$. Similarly, for $f \in H^1(a, b)$, $y' \in H^1(a, b)$ and $y'' \in L^2(a, b)$ so that y $\in H^2(a, b)$. Any result which describe the smoothness of solution depending on the smoothness of given data is called regularity result.

Theorem 1.3.1. *For $f \in L^2(\Omega)$, the problem (1.4) has unique solution $u \in H^2(\Omega) \cap H_0^1(\Omega)$ satisfying following a priori estimate*

$$\|u\|_{H^2(\Omega)} \leq C\|f\|_{L^2(\Omega)}$$

1.4 Embedding Result

The following result connects Sobolev space and classical function spaces, and thus concludes this chapter. For our convenience, we will now state the Sobolev embedding theorem and Rellich's theorem. It's reasonable to wonder if members of $H^m(\Omega)$ are just functions that are continuous as well as their derivatives of order $\leq$ m $-$ 1. After all, it's not easy to picture a non-continuous function in $H^1(\Omega)$, for example. Sobolev's famous theorem states that, as expected, all members of $H^1(a, b)$ are continuous functions, but that this is not true for higher dimensional domains. Such results are called Embedding results.

Let X, Y are Banach Spaces with X $\subseteq$ Y . We say that X is continuously embedded in Y i.e. X $\to$ Y if the identity map I : X $\to$ Y is continuous that is

$$\|I(x)\|_Y \leq C\|x\|_X \ \textit{for some constant } C > 0$$

For Sobolev spaces to be embedded in the space of continuous functions certain conditions need to be met. The following theorem describes those conditions.

Theorem 1.4.1. The Sobolev Embedding Theorem. *Let Ω be a bounded domain in R^n with Lipschitz boundary Γ. If $m - k > n/2$, then every function in $H^m(\Omega)$ belongs to $C^k(\bar{\Omega})$. Furthermore, the embedding*

$$H^m(\Omega) \subset C^k(\bar{\Omega}) \tag{1.5}$$

is continuous.

The aforementioned finding must be interpreted with caution. Remember that $H^m(\Omega)$ members are function equivalence classes since they are L^2 members, whereas continuous functions are clearly specified. As a result,

the embedding (1.5) must be interpreted in the sense that each member of $H^m(\Omega)$ can be linked to a $C^k(\bar{\Omega})$ function, possibly after changing the function's values on a set of measure zero.

According to the Sobolev Embedding Theorem, the functions in $H^1(\Omega)$ are continuous if n = 1 and Ω is a subset of the real line. However, we require that a function be a member of $H^2(\Omega)$ in order to assure the continuity of functions with domains that are subsets of the plane.

Chapter 2

Computational Aspects of FEM

The discretization of the domain into elements is discussed in this chapter, followed by the construction of a finite element space. Finally, we look at evaluating the model using reference elements on a simplified domain.

2.1 Finite Element Discretization

The first step towards FEM is to discretize the computational domain $\Omega \in R^2$ into finitely many elements.

Definition 2.1.1. Triangulation A triangulation T_h is a partition of $\bar{\Omega}$ into a finite number of triangular subsets K for splitting the domain into discrete pieces, such that:

1. $\bar{\Omega} = \cup_{K \in T_h} K$
2. $K = \overline{K}$ and $int(K) \neq \phi$ where $K \in T_h$
3. If $K_1, K_2 \in T_h$ and $K_1 \neq K_2$ then either $K_1 \cap K_2 = \phi$ or $K_1 \cap K_2$ is a common vertex or edge of K_1 and K_2. This implies that no vertex of any triangle can lie on the interior of an edge of another triangle.

4. For any triangle $K \in T_h$, let $r_k, \overline{r_k}$ be the radii of the inscribed and the circumscribed circles of K respectively. Let $h = max\{\overline{r_k} : K \in T_h\}$. We assume that for some fixed $h_0 > 0$, there exist two positive constants C_0 and C_1 such that

$$C_0 h \leq diam(K) \leq C_1 h \quad \forall K \in T_h, \ \forall h \in (0, h_0)$$

We construct our finite element space V_h, which is finite dimensional and piece-wise polynomial, based on the triangulation T_h. Over the elements K, we create polynomial spaces.

Definition 2.1.2. The triple $(K, \ P_r, \ \Sigma_K)$ is called a finite elements. Σ_K represents the set of conditions such that unique determination of any $p \in P_r$ is possible.

Lagrange Finite Elements:

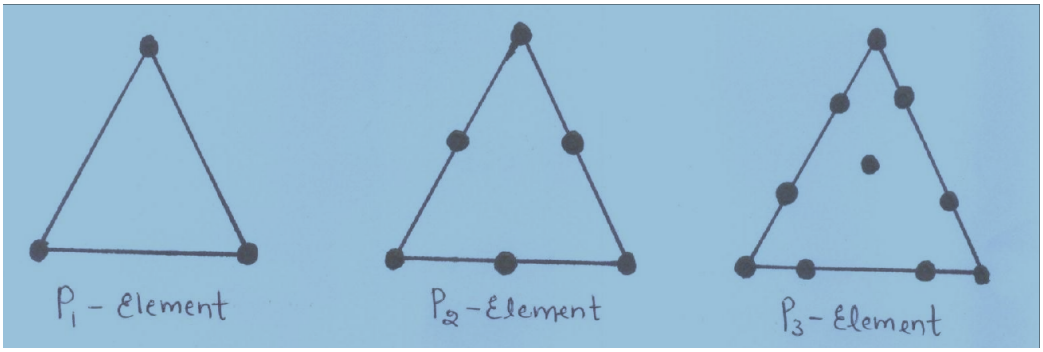

Figure 2.1: Different Lagrange Elements

For given P_r we consider polynomial space of degree $\leq \ r$. In $\mathbb{R}^2$,

$$dim(P_r) \ = \ \frac{(r+1)(r+2)}{2} \ = \ n_r$$

.

We will explore linear P_1 and quadratic P_2 elements as examples of lagrange elements, as well as the building of finite element space for the elements. Let us begin by introducing the nodal basis function.

Definition 2.1.3. Nodal Basis Function: Firstly, let $N_h(\bar{\Omega}) = \{x_i \ : 1 \leq i \leq n_h\}$ denote set of all nodes generated by triangulation. Then, $\phi_i \ \forall(1 \leq i \leq n_h)$ such that it satisfies $\phi_i(x_j) = \delta_{ij}$ is called the nodal basis function.

Construction of (K, P_1, Σ_K)

This utilizes area of triangles as the factor for the basis functions. Consider, element K is a triangle with vertices (nodes) $a_1 = A$, $a_2 = B$, $a_3 = C$. Further assume, P to be arbitrarily located in K. Now, corresponding to $a_1 = A$, let us define:

$$\phi_1(P) \ = \ \frac{Area(\Delta PBC)}{Area(\Delta ABC)}$$

The same can be done corresponding to points $a_2 = B$ and $a_3 = C$:

$$\phi_2(P) \ = \ \frac{Area(\Delta PCA)}{Area(\Delta ABC)}$$

$$\phi_3(P) \ = \ \frac{Area(\Delta PAB)}{Area(\Delta ABC)}$$

We can observe that $\phi_1(x) \ + \phi_2(x) \ + \phi_3(x) \ = \ 1$.

More importantly, for the P_1 element and $N_h(K) = \{a_1, a_2, a_3\}$, we have ϕ_i as the nodal basis function, since $\phi_i(a_j) = \delta_{ij}$.

Construction of (K, P_2, Σ_K)

The basis functions calculated for the P_1 element can be extended as below. We have, $N_h = \{a_1, a_2, a_3, a_{12}, a_{23}, a_{31}\}$. The basis function ϕ_i corresponding

to node a_i is defined as:

$$\phi_i(x) = \lambda_i(x)(2\lambda_i(x) - 1) \qquad \forall\ 1 \leq i \leq 3$$

The basis function corresponding to a_{ij} can be defined as below:

$$\phi_{ij}(x) = 4\lambda_i(x)\lambda_j(x) \qquad \forall\ 1 \leq i, j \leq 3$$

Here, λ_i represents the nodal basis function for the P_1 element.
Now, the finite element space is defined as

$$V_h = \{v_h : \bar{\Omega} \to \mathbb{R} : v_{h|K} \in P_r(K), K \in T_h\}.$$

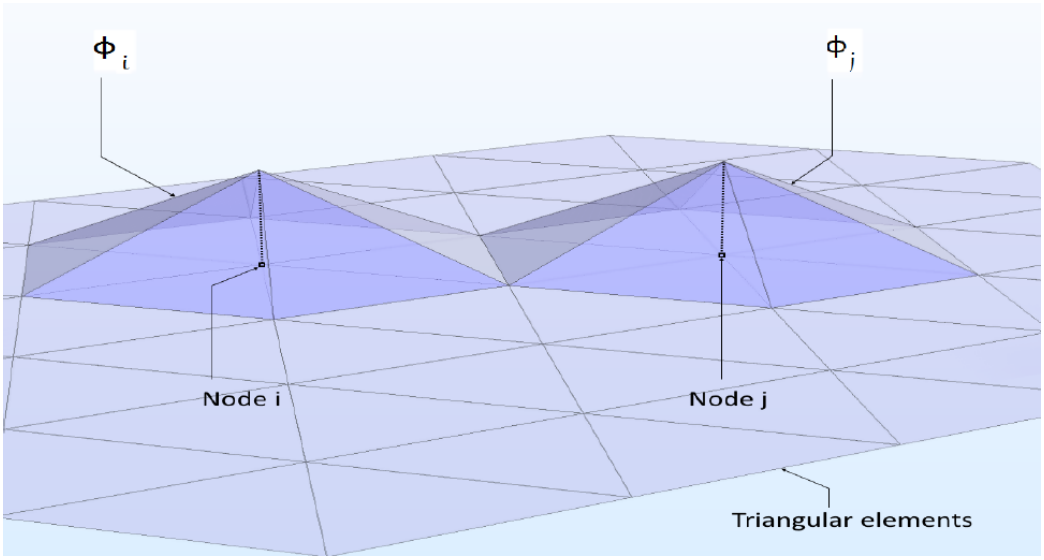

Figure 2.2: Nodal Basis Function for Lagrange Element

Based on the finite element space V_h, two categories of FEM can be defined. If $V_h \in H$ it is known as conforming FEM, whereas if $V_h \notin H$ it is known

as non-conforming FEM. Here, H represents the solution space for the given BVP. This project only deals with conforming FEM.

2.2 Reference Element

Using the weak formulation of the BVP and the finite elements we generated above, we can proceed to find the solution $u_h \in V_h$ such that

$$A(u_h, v_h) = L(v_h) \quad \forall\ v_h \in V_h$$

Many calculations need to be done separately for each triangular element K. To avoid repeating similar calculations for different elements and to make the algorithm more efficient, the Reference Element Technique is used.

Definition 2.2.1. Reference Triangle: Any arbitrary triangle $K \in \mathbb{R}^2$ can be represented as an image of the unit triangle $\hat{K}$ under a map:

$$F_k : \mathbb{R}^2 \to \mathbb{R}^2 \ i.e.\ \hat{x} \to F_K(\hat{x}) = B_K(\hat{x}) + b_K$$

where $B_K \in \mathbb{R}^2 \times \mathbb{R}^2$ represents a non-singular matrix and $b_K \in \mathbb{R}^2$. The unit triangle $\hat{K}$ is referred to as the reference triangle.

Let $\hat{K}$ be the triangle ΔABC, where $a_1 = (0,0) = A,\ a_2 = (1,0) = B,\ a_3 = (0,1) = C$. The nodal basis functions over $\hat{K}$ can be given by:

$$\phi_1(x,y) = 1 - x - y,\ \phi_2(x,y) = x,\ \phi_3(x,y) = y$$

Suppose $K \in T_h$ is element with vertices $P_1^K = (x_1, y_1),\ P_2^K = (x_2, y_2),\ P_3^K = (x_3, y_3)$. Here, we can define $F_K(\hat{X}) = P_1^K \phi_1(\hat{X}) + P_2^K \phi_2(\hat{X}) + P_3^K \phi_3(\hat{X})$.

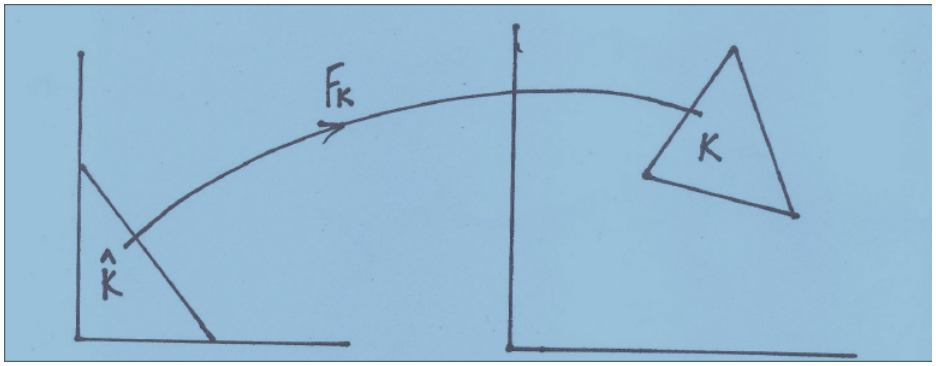

Figure 2.3: Reference Element

We obtain that the nodes of the reference triangle $\hat{K}$ map to the nodes of our given triangle K. Also, the map F_K is well defined so that each $\hat{X} \in \hat{K}$ maps to a unique $X \in K$. We obtain

$$F_K(\hat{X}) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Here, $B_K = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}$ which we can verify easily to be non-singular. Also, $b_K = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$.

We get $\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = B_K^{-1} \begin{bmatrix} x - x_1 \\ y - y_1 \end{bmatrix}$. So we conclude that F_K^{-1} exists.

Further let $N_h(K) = \{x_1, x_2, x_3, \cdots, x_{nr}\}$ denote the set of nodes for K, and let ϕ_i denote the nodal basis functions. We can now observe,

$$x_i = F_K(\hat{x}_i)$$

$$\phi_i(x_j) \;=\; \delta_{ij} \;=\; \hat{\phi}_i(\hat{x}_j) \;=\; \hat{\phi}_i \cdot F_K^{-1}(x_j)$$

$$\therefore\; \phi_i = \hat{\phi}_i \cdot F_K^{-1}$$

Making use of the reference element technique, computation becomes much simpler since all the computations need to be made only on the reference element, which can then be transformed to the required element.

Chapter 3

Weak Galerkin FEM

The classical finite element methods, based on conforming finite element discretization, have limitations in practical computation. The conforming finite element space is restricted to piecewise polynomials with prescribed continuity that ensures conformity and stability of the corresponding weak formulation. Keeping in mind the applicability of polygonal meshes, recently attempts have been made to develop certain technologies which make use of polygonal meshes. Due to the use of discontinuous approximation functions, WG-FEMs are highly flexible in construction of finite element spaces of any orders with the price of more degrees of freedom and more complex formulations.

A typical local WG element is of the form $(\mathcal{P}_k(K),\ \mathcal{P}_j(\partial K),\ \left[\mathcal{P}_l(K)\right]^2)$, where $k \geq 1$ is the degree of polynomials in the interior of the element K, $j \geq 0$ is the degree of polynomials on the boundary of K, and $l \geq 0$ is the degree of polynomials employed in the computation of weak gradients or weak first order partial derivatives.

3.1 Weak Galerkin Methods

Weak Problem: We recall following weak formulation to the BVP: Find $u \in H_0^1(\Omega)$ such that

$$\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx \ \ \forall v \in H_0^1(\Omega). \tag{3.1}$$

The WG finite element formulations can be derived from the variational form by replacing differential operator involved by weak derivatives. More precisely, the classical gradient in (3.1) is replaced by *weak gradient*, which is discussed below.

Motivation: Weak Function

- Let K be any polygonal or polyhedral domain with interior K^0 and boundary ∂K.
- A weak function on the region K refers to a pair of scalar-valued functions $v = \{v_0, v_b\}$ such that $v_0 \in L^2(K)$ and $v_b \in H^{\frac{1}{2}}(\partial K)$.
- Note that v_b may not be necessarily related to the trace of v_0 on ∂K. Denote by $\mathcal{V}(K)$ the space of weak functions on K; *i. e.*,

$$\mathcal{V}(K) = \{v = \{v_0, v_b\} : v_0 \in L^2(K), v_b \in H^{\frac{1}{2}}(\partial K)\}. \tag{3.2}$$

- For any weak function $v = \{v_0, v_b\}$, its weak gradient $\nabla_w v$ is defined (interpreted) as a linear functional on $H(\mathrm{div}, K)$ whose action on each $q \in H(\mathrm{div}, K)$ is given by

$$(\nabla_w v, q)_K = -\int_K v_0 \nabla \cdot q dK + \int_{\partial K} v_b q \cdot \mathbf{n} ds, \tag{3.3}$$

  where $\mathbf{n}$ is the outward normal to ∂K.

Weak Galerkin Space $\left(\mathcal{P}_k(K),\ \mathcal{P}_j(\partial K),\ \left[\mathcal{P}_l(K)\right]^2\right)$**:**

- $k \geq 1$ is the degree of polynomials in the interior of the element K
- $j \geq 0$ is the degree of polynomials on the boundary of K and
- $l \geq 0$ is the degree of polynomials employed in the computation of weak gradients or weak first order partial derivatives.
- The accuracy and the computational complexity of the corresponding WG scheme is significantly impacted by the selection of such polynomials.
- $k,\ j,\ l$ are selected in such a way that minimizes the number of unknowns in the numerical scheme without compromising the accuracy of the numerical approximation. A lowest order WG-FEM space is $\left(\mathcal{P}_1(K),\ \mathcal{P}_0(\partial K),\ \left[\mathcal{P}_0(K)\right]^2\right)$.

Weak Galerkin Approximation:

For $k \geq 1$, let V_h be a WG FE space associated with $\mathcal{T}_h$ defined as

$$V_h(k,k) = \{v = \{v_0, v_b\} :\ v_0|_{K^0} \in \mathcal{P}_k(K),\ v_b|_e \in \mathcal{P}_k(e),\ e \in \partial K,\ K \in \mathcal{T}_h\}.$$

- Note that functions in V_h are defined on each element, and there are two-sided values of v_b on each interior edge/face e, depicted as $v_b|_{\partial T_1}$ and $v_b|_{\partial T_1}$ in above Figure.
- We assume that v_b has a unique value on each interior edge/face e, that is

$$[v]_e = 0 \quad \forall e \in \mathcal{E}_h^0,$$

where $[v]_e$ denotes the jump of $v \in V_h$ across an interior edge $e \in \mathcal{E}_h^0$.

- We write $V_h^0 = \{v = \{v_0, v_b\} \in V_h : \ v_b = 0 \ \text{ on } \partial\Omega\}$.

- For $v \in V_h$, the discrete weak gradient of it is defined as the unique polynomial $(\nabla_w v) \in [\mathcal{P}_{k-1}(K)]^2$ that satisfies the following equation

$$(\nabla_w v, \phi)_K = -\int_K v_0(\nabla \cdot \phi)dK + \int_{\partial K} v_b(\phi \cdot \mathbf{n})ds \ \ \forall \phi \in [\mathcal{P}_{k-1}(K)]^2. \tag{3.4}$$

- Our weak Galerkin space is $\big(\mathcal{P}_k(K),\ \mathcal{P}_k(\partial K),\ \big[\mathcal{P}_{k-1}(K)\big]^2\big)$.

Weak Galerkin Method for General Partitions:

- We define a bilinear map $a(\cdot,\cdot) : V_h \times V_h \to \mathbb{R}$ by

$$a(u_h, v_h) = (\alpha \nabla_w u_h, \nabla_w v_h) = \sum_{K \in \mathcal{T}_h} (\alpha \nabla_w u_h, \nabla_w v_h)_K, \tag{3.5}$$

- with a stabilizer $s(\cdot,\cdot) : V_h \times V_h \to \mathbb{R}$ defined by

$$s(u_h, v_h) = \sum_{K \in \mathcal{T}_h} h_K^{-1} \langle u_0 - u_b, v_0 - v_b \rangle_{\partial K}. \tag{3.6}$$

- **Weak Galerkin Algorithm:** Find $u_h = \{u_0, u_b\} \in V_h^0$ such that

$$a_s(u_h, v_h) = (f, v_h) \ \ \forall v_h \in V_h^0, \tag{3.7}$$

  with

$$a_s(u_h, v_h) = a(u_h, v_h) + s(u_h, v_h). \tag{3.8}$$

- It is important to check that $a_s(\cdot,\cdot)$ is positive so that WG approximation has a unique solution.

L^2-Projections:.

- For each element $K \in \mathcal{T}_h$, denote by Q_0 the usual L^2 projection operator from $L^2(K)$ onto $\mathcal{P}_k(K)$ and by Q_b the L^2 projection from $L^2(e)$ onto $\mathcal{P}_{k-1}(e)$ for any $e \in \mathcal{E}_h$.

- We shall combine Q_0 with Q_b by writing $Q_h = \{Q_0, Q_b\}$. More precisely, for $\phi \in H^1(K)$, we have $Q_h\phi = \{Q_0\phi, Q_b\phi\}$.

- In addition to Q_h, let $\mathbb{Q}_h$ be an another local L^2 projection from $[L^2(K)]^2$ onto $[\mathcal{P}_{k-1}(K)]^2$.

Error Equation:

- As in finite element method, we split our error into two components using an intermediate operator. We write

$$u - u_h = (u - Q_h u) + (Q_h u - u_h).$$

- For simplicity, we introduce the following notation

$$e_h := \{e_0, e_b\} = u_h - Q_h u. \tag{3.9}$$

- Testing original equation by using v_0 of $v = \{v_0, v_b\} \in V_h^0$, we arrive at

$$a_s(Q_h u, v) = (f, v_0) + s(Q_h u, v) - \langle v_0 - v_b, (\mathbb{Q}_h(\nabla u) - \nabla u) \cdot \mathbf{n} \rangle_{\partial K} \tag{3.10}$$

- Finally, subtracting WG approximation (3.7) from (3.10), we obtain

$$a_s(e_h, v) = l_1(u, v) + l_2(u, v) \ \ \forall v \in V_h^0, \tag{3.11}$$

where bilinear forms $l_1(\cdot,\cdot)$ and $l_2(\cdot,\cdot)$ are given by

$$\begin{aligned} l_1(u,v) &= \sum_{K\in\mathcal{T}_h} \langle (\nabla u - \mathbb{Q}_h(\nabla u))\cdot \mathbf{n}, v_0 - v_b\rangle_{\partial K}, \\ l_2(u,v) &= \sum_{K\in\mathcal{T}_h} h_K^{-1}\langle Q_0 u - Q_b(u|_{\partial K}), v_0 - v_b\rangle_{\partial K}. \end{aligned}$$

Convergence Results:

- **Convergence Results for H^1-norm:** Let $u_h \in V_h$ be the weak Galerkin finite element solution of the problem (3.7). Assume that the exact solution is so regular that $u \in H^{k+1}(\Omega)$. Then, there exists a constant C such that

$$\|e_h\| \leq Ch^k \|u\|_{k+1,\Omega}.$$

Now, for L^2 norm error estimate, duality argument leads to following convergence result.

- **Convergence Results for L^2-norm:** Let $u_h \in V_h$ be the weak Galerkin finite element solution of the problem (3.7). Assume that the exact solution is so regular that $u \in H^{k+1}(\Omega)$. Then, there exists a constant C such that

$$\|e_0\| \leq Ch^{k+1} \|u\|_{k+1,\Omega}. \quad \square$$

Chapter 4

Mesh Generation

The first step in FEM is generating a mesh of elements to work upon. We first look at uniform triangulation of a rectangular domain, followed by mesh generation on a cubical domain. We shall then look at extracting important information about the elements generated above.

4.1 Rectangular Domain

4.1.1 Mesh Generation

When considering mesh generation on a two dimensional domain, the simplest form is triangulation of a rectangular domain. The following algorithm generates triangular elements from a given rectangular domain with a fixed step size in both directions. Figure 4.1 shows a rectangular domain defined by $[-1, 1] \times [-1, 1]$. It has been triangulated with a step size of 1 in both directions. Each node and element has been numbered.

Once the mesh has been generated, we follow it up by extracting all the necessary information about the elements and their positioning.

Algorithm 1 Triangulation of Rectangular Domain

Require: $[x_0, x_1, y_0, y_1]$ denoting the domain and *step size* denoting the step size

Ensure: node contains array of all nodes. elem contains list of all elements.

$square \leftarrow [x_0, x_1, y_0, y_1]$

$h \leftarrow step\ size$

▷ Creating Array of Nodes

$x_0 \leftarrow square(1), x_1 \leftarrow square(2), y_0 \leftarrow square(3), y_1 \leftarrow square(4)$

$h_x \leftarrow h(1), h_y \leftarrow h(2)$

Generate meshgrid from $[x_0 : h_x : x_1] \times [y_0 : h_y : y_1]$

node $\leftarrow$ array of vertices from meshgrid

▷ Listing Elements

$ni \leftarrow$ number of rows in mesh

$k \leftarrow 1$

while $k \leq$ Number of nodes $- ni$ **do**

$elem \leftarrow [elem, (k, k + ni, k + ni + 1)]$

$elem \leftarrow [elem, (k, k + 1, k + ni + 1)]$

$k \leftarrow k + 1$

end while

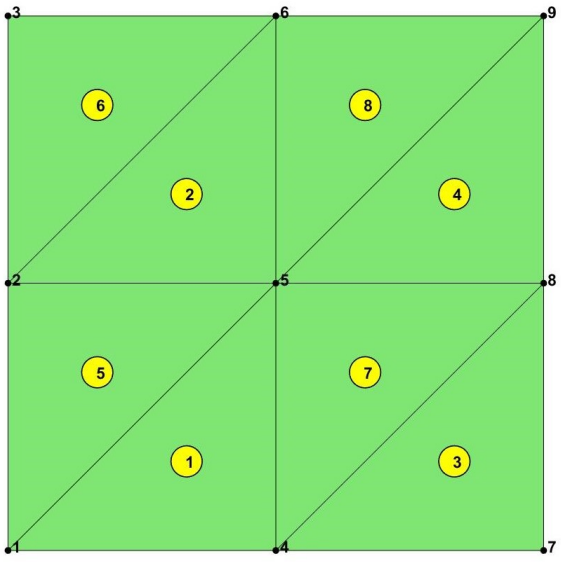

Figure 4.1: Mesh Generation in a Rectangular Domain

4.1.2 Information Extraction

Before we can begin extracting the mesh information, we need to decide what all information we shall need in the future for making the FEM solver.
We will obviously need to recognize the edges and what element they are a part of. Since, for solving any PDE, we will have boundary conditions which will have to be satisfied, therefore we will need to mark all the edges and elements which lie on the boundary of our mesh. Further, we will also require information about neighboring elements. We will now proceed to extract all of this from the list of nodes and elements we generated earlier.

```
function T = auxstructure(elem)
%  AUXSTRUCTURE auxiliary structure for a 2-D triangulation.
%  T = AUXSTRUCTURE(elem) constucts the indices map between elements, edges
%  and nodes, and the boundary information. T is a structure.

    totalEdge = uint32(sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])], 2));
    [edge,i_tot_last,i_edge] = myunique(totalEdge);
    NT = size(elem, 1);
    elem2edge = uint32(reshape(i_edge, NT, 3));
    i_tot_first(i_edge(3*NT:-1:1)) = 3*NT:-1:1;
    i_tot_first = i_tot_first';
    first_elem_edge = ceil(i_tot_first/NT);
    last_elem_edge = ceil(i_tot_last/NT);
    first_elem = i_tot_first - NT*(first_elem_edge-1);
    last_elem = i_tot_last - NT*(last_elem_edge-1);
    diff_ind = (i_tot_first ~= i_tot_last);
    neighbor = uint32(accumarray([[first_elem(diff_ind) first_elem_edge(diff_ind)];...
        [last_elem last_elem_edge]], [last_elem(diff_ind); first_elem], [NT 3]));
    edge2elem = uint32([first_elem last_elem first_elem_edge last_elem_edge]);
    bdElem = first_elem(first_elem == last_elem);
    first_edge_bd = first_elem_edge(first_elem == last_elem);
    bdEdge = uint32([elem(bdElem(first_edge_bd==1),[2 3]);...
        elem(bdElem(first_edge_bd==2),[3 1]); elem(bdElem(first_edge_bd==3),[1 2])]);
    bdEdge2elem = uint32([bdElem(first_edge_bd==1);bdElem(first_edge_bd==2);...
        bdElem(first_edge_bd==3)]);
    T = struct('neighbor', neighbor, 'edge2elem', edge2elem, 'edge', edge,...
        'elem2edge', elem2edge, 'bdElem', bdElem, 'bdEdge', bdEdge,...
        'bdEdge2elem', bdEdge2elem);
end
```

From the list of elements that we have, we can very easily construct a list of edges as well as a map from edge to respective element and vice versa. Further, we notice that each internal edge appears twice while counting, as we account for each element. Whereas, the boundary edges appear only once. Also, elements which share an edge between them define a pair of neighboring elements. We have used these observations in the code above to extract lists of edges, boundary elements, boundary edges, neighbouring elements and a map between edges and elements. All of this information has been stored in a structure T.

4.2 Cubical Domain

4.2.1 Mesh Generation

Similar to the two dimensional domain, we have taken into consideration a simple form of mesh generation on a cubical domain. In place of triangles, we consider each element to be a tetrahedron. The following code generates tetrahedrons of fixed step size in all three directions from a given cuboid in the three dimensional space. Figure 4.2 shows a cubical domain defined by $[0, 1] \times [0, 1] \times [0, 1]$, which has been divided into six tetrahedrons with step size 1 in each direction. We can see that with smaller step sizes, each cube/cuboid formed would be divided into six tetrahedrons each. In the figure each node and each element has been numbered.

```
function [node, elem, HB] = cubemesh(box,h)
    x0 = box(1); x1 = box(2);
    y0 = box(3); y1 = box(4);
    z0 = box(5); z1 = box(6);
    if length(h) == 1
        [x, y, z] = ndgrid(x0:h:x1, y0:h:y1, z0:h:z1);
    elseif length(h) == 3
        [x, y, z] = ndgrid(x0:h(1):x1, y0:h(2):y1, z0:h(3):z1);
    end
    node = [x(:), y(:), z(:)];
    [nx, ny, nz] = size(x);
    elem = zeros(6*(nx-1)*(ny-1)*(nz-1), 4);
    indexMap = reshape(1:nx*ny*nz,nx,ny,nz);
    localIndex = zeros(8,1);
    idx = 1;
    for k = 1:nz-1
        for j = 1:ny-1
            for i = 1:nx-1
                localIndex(1) = indexMap(i,j,k);
                localIndex(2) = indexMap(i+1,j,k);
                localIndex(3) = indexMap(i+1,j+1,k);
                localIndex(4) = indexMap(i,j+1,k);
                localIndex(5) = indexMap(i,j,k+1);
                localIndex(6) = indexMap(i+1,j,k+1);
                localIndex(7) = indexMap(i+1,j+1,k+1);
                localIndex(8) = indexMap(i,j+1,k+1);
                elem(idx:idx+5,:) = localIndex([1 2 3 7; 1 4 3 7; 1 5 6 7;...
                                                1 5 8 7; 1 2 6 7; 1 4 8 7]);
                idx = idx + 6;
            end
        end
    end
    N0 = size(node,1);
    HB = zeros(N0,4);
    HB(1:N0,1:3) = repmat((1:N0)',1,3);
end
```

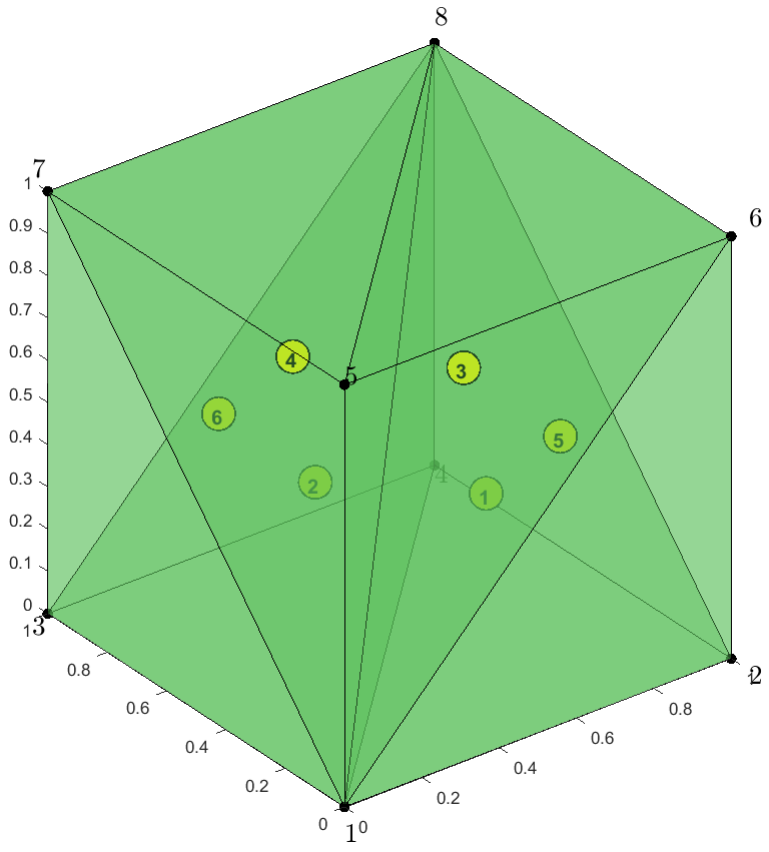

Figure 4.2: Mesh Generation in a Cubical Domain

4.2.2 Information Extraction

We will extract the same set of information as in the two dimensional case. However, we are now interested in faces instead of edges. Our set of observations for writing the code are also all similar to the rectangular case. The following code has been used for extracting information about the tetrahedral elements.

```
function T = auxstructure3(elem)
%%  AUXSTRUCTURE3 auxstructure for 3 dimensional elements
%   T = suxstructure3(elem) makes an indices map between the elements, faces,
%   and nodes and the boundary information. T is a structure.

    totalFace = uint32(sort([elem(:,[2 3 4]); elem(:,[1 4 3]);...
        elem(:,[1 2 4]); elem(:,[1 3 2])], 2));
    [face, i_tot_last, i_face] = myunique(totalFace);
    NT = size(elem, 1);
    elem2face = uint32(reshape(i_face, NT, 4));
    i_tot_first(i_face(4*NT:-1:1)) = 4*NT:-1:1;
    i_tot_first = i_tot_first';
    first_elem_face = ceil(i_tot_first/NT);
    last_elem_face = ceil(i_tot_last/NT);
    first_elem = i_tot_first - NT*(first_elem_face-1);
    last_elem = i_tot_last - NT*(last_elem_face-1);
    diff_ind = (i_tot_first ~= i_tot_last);
    neighbor = uint32(accumarray([[first_elem(diff_ind) first_elem_face(diff_ind)];...
        [last_elem last_elem_face]], [last_elem(diff_ind); first_elem], [NT 4]));
    face2elem = uint32([first_elem last_elem first_elem_face last_elem_face]);
    bdElem = first_elem(first_elem == last_elem);
    first_face_bd = first_elem_face(first_elem == last_elem);
    bdFace = uint32([elem(bdElem(first_face_bd==1),[2 3 4]);...
                     elem(bdElem(first_face_bd==2),[1 3 4]);...
                     elem(bdElem(first_face_bd==3),[1 2 4]);...
                     elem(bdElem(first_face_bd==4),[1 3 2])]);
    bdFace2elem = uint32([bdElem(first_face_bd==1);bdElem(first_face_bd==2);...
        bdElem(first_face_bd==3);bdElem(first_face_bd==4)]);
    T = struct('neighbor',neighbor,'elem2face',elem2face,'face',uint32(face),...
        'face2elem',face2elem,'bdElem',bdElem,'bdFace',bdFace,'bdFace2elem', bdFace2elem);
end
```

We have used this code to extract lists of faces, boundary elements, boundary faces, neighboring elements and a map between faces and elements. All of this has been stored in a structure T.

Chapter 5

FEM Algorithm

Now that we have the mesh structure ready for the cubical domain, we can proceed towards making an FEM solver. We have considered the Poisson Equation as the example. In the previous chapters, we have established the variational formulation for the Poisson Equation, nodal basis functions, finite element discretization and reference element technique. We will utilize all of this and put it together with the mesh generation code to obtain a functional FEM solver for the Poisson Equation.

Our model problem is as defined in Equation 1.1,

$$-\Delta u = f \ in \ \Omega$$
$$u = 0 \ \ on \ \delta\Omega$$

The variational formulation for the same is as in Equation 1.4,

$$\int_\Omega \nabla u.\nabla v dx = \int_\Omega f v dx \quad \forall v \in H_0^1(\Omega)$$

5.1 Boundary Conditions and Nodal Basis Function

Here, we need to notice that the value of our function of interest, u, is zero on the entire boundary $\delta\Omega$. To approximate u, we represent it as the linear combination of the nodal basis functions. Hence, we take all the basis functions defined on the boundary nodes and faces to be identically zero.

5.1.1 Removing the Boundary

Since only the internal nodes are of interest to us, we need a way to distinguish between the nodes on the boundary and those in the interior of our domain. Recall that we had stored the list of the faces on the boundary in the structure T during mesh generation. We can make use of that to mark the interior nodes. The following function does the same and returns two boolean arrays. The arrays indicate the indices of the nodes and the faces which lie in the interior.

```
function [intNodeIndex, intFaceIndex] = remove_bdNodeFace(node, T)
    bdFace = double(T.bdFace);
    [bdNode, ~, ~] = myunique(bdFace(:));
    N = size(node, 1);
    allNode = (1:N)';
    intNodeIndex = allNode(~ismember(allNode, bdNode));
    bdFace = sort(bdFace, 2);
    N = size(T.face, 1);
    allFace = (1:N)';
    intFaceIndex = allFace(~ismember(T.face, bdFace, 'rows'));
end
```

Now that we have removed the boundary, we can proceed towards defining the basis functions.

5.1.2 Defining Basis Functions

We will use the Reference Element Technique as described in Section 2.2 to define the basis functions on each node. Therefore, to begin with, let us consider the unit tetrahedron defined by the points $P_1 = (0, 0, 0),\ P_2 = (1, 0, 0),\ P_3 = (0, 1, 0),\ P_4 = (0, 0, 1)$.

We define the basis functions for these four nodes of the reference tetrahedron. We take $\phi_{P_1}(x, y, z) = 1 - x - y - z$, $\phi_{P_2}(x, y, z) = x$, $\phi_{P_3}(x, y, z) = y$ and $\phi_{P_4}(x, y, z) = z$. All these functions take value zero at all points outside the element.

We will extend the definition of the map F_K, as defined in Section 2.2, from two dimensions to three dimensions. Therefore, we have, $F_K(\hat{X}) = P_1^K\phi_1(\hat{X}) + P_2^K\phi_2(\hat{X}) + P_3^K\phi_3(\hat{X}) + P_4^K\phi_4(\hat{X})$. Now, we can define the basis functions for any element K as $\phi_i(X) = \hat{\phi}_i(F_K^{-1}(X))$.

We already know from earlier that F_K^{-1} exists. Also,

$$F_K^{-1}(\begin{bmatrix} x \\ y \\ z \end{bmatrix}) = B_K^{-1} \begin{bmatrix} x - x_1 \\ y - y_1 \\ z - z_1 \end{bmatrix}$$

$$B_K = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{bmatrix}$$

This will be useful when we need to calculate the value of $\nabla\phi_i$.

5.2 Solving Variational Formulation

$$\int_\Omega \nabla u.\nabla v dx \ = \int_\Omega f v dx \quad \forall v \in H_0^1(\Omega)$$

Here, we have already taken u as the linear combination of the nodal basis functions. Since the above equation is valid for all $v \in H_0^1(\Omega)$, we can take v to be the basis functions. Thus, we shall have as many equations as the unknown coefficients. We can break down the above computation into two parts:

- Calculating the RHS
- Calculating the LHS

5.2.1 Calculating the RHS

As we attempt to approximate the RHS, there are a few observations that we need to make. What we are essentially calculating is $\int_\Omega f\phi_i dx$. Each ϕ_i is defined on some node such that it takes value zero outside of the elements that the node is a part of. Each internal node is a part of 24 surrounding tetrahedral elements. We can now proceed to approximate the integral over some element T with nodes P_1, P_2, P_3, P_4.

$$\int_T f\phi_i dx \approx vol(T) * \frac{f(P_1)\phi_i(P_1) + f(P_2)\phi_i(P_2) + f(P_3)\phi_i(P_3) + f(P_4)\phi_i(P_4)}{4}$$

We know that volume of a tetrahedron is one-sixth that of the cuboid it is a part of. Also, for any node a_j, $\phi_i(a_j) = \delta_{ij}$. Using all of these observations, we get

$$\int_\Omega f\phi_i dx \approx h_x h_y h_z f(P)$$

P is the node on which ϕ_i has been defined and h_x, h_y, h_z denote the step sizes in the respective directions.

```
function L = RHS(node, intNodeIndex, h)
    intNode = node(intNodeIndex, :);
    if size(h) == 1
        h = [h, h, h];
    end
    L = f(intNode(:, 1), intNode(:, 2), intNode(:, 3))*h(1)*h(2)*h(3);
end
```

The above function calculates the RHS of our variational formulation as described above.

5.2.2 Calculating the LHS

Calculation of the LHS will involve calculating $\int_T \nabla\phi_i.\nabla\phi_j$. As established earlier, $\phi(X) = \hat{\phi}(F^{-1}(X))$. Therefore, we have,

$$\begin{bmatrix} \frac{\delta\phi}{\delta x} & \frac{\delta\phi}{\delta y} & \frac{\delta\phi}{\delta z} \end{bmatrix} = \begin{bmatrix} \frac{\delta\hat{\phi}}{\delta x} & \frac{\delta\hat{\phi}}{\delta y} & \frac{\delta\hat{\phi}}{\delta z} \end{bmatrix} \begin{bmatrix} \frac{\delta F_1^{-1}}{\delta x} & \frac{\delta F_1^{-1}}{\delta y} & \frac{\delta F_1^{-1}}{\delta z} \\ \frac{\delta F_2^{-1}}{\delta x} & \frac{\delta F_2^{-1}}{\delta y} & \frac{\delta F_2^{-1}}{\delta z} \\ \frac{\delta F_3^{-1}}{\delta x} & \frac{\delta F_3^{-1}}{\delta y} & \frac{\delta F_3^{-1}}{\delta z} \end{bmatrix}$$

Let us notice, that the Jacobian J of F^{-1} used above is equal to the matrix B_K^{-1} from Section 5.1.2. Moreover, since we defined $\hat{\phi}$ explicitly on the reference element, we know the value of $\nabla\hat{\phi}$ and can in turn find the value of $\nabla\phi_i$.

Once more, we need to keep in mind that the basis functions take non zero value only in the elements that they belong to, outside of which their value is zero. Hence, the dot product, $\nabla\phi_i.\nabla\phi_j$, in the LHS will hold a non zero value only when both ϕ_i and ϕ_j belong to the same element. Further, we will

approximate the integral in the same manner as we did for the RHS. The following function calculates the LHS and returns the coefficient matrix A.

```
function A = LHS(node,elem,intNodeIndex,h)
    N = size(node, 1);
    intFlag = ismember(1:N, intNodeIndex);
    itgrl = sparse(N, N);
    dphi = [1,0,0;0,1,0;0,0,1;-1,-1,-1];
    for i = 1:size(elem, 1)
        loc = elem(i, :);
        F = [node(loc(1),1)-node(loc(4),1),node(loc(2),1)-node(loc(4),1),..
            node(loc(3),1)-node(loc(4),1); node(loc(1),2)-node(loc(4),2),..
            node(loc(2),2)-node(loc(4),2),node(loc(3),2)-node(loc(4),2);...
            node(loc(1),3)-node(loc(4),3),node(loc(2),3)-node(loc(4),3),...
            node(loc(3),3)-node(loc(4),3)];
        for j = 1:4
            if ~intFlag(loc(j))
                continue;
            end
            for k = 1:4
                if ~intFlag(loc(k))
                    continue;
                end
                dphij = dphi(j,:)/F;
                dphik = dphi(k,:)/F;
                calc = dot(dphij, dphik)*h(1)*h(2)*h(3)/6;
                itgrl(loc(k), loc(j)) = itgrl(loc(k), loc(j)) + calc;
            end
        end
    end
    A = itgrl(intNodeIndex,intNodeIndex);
end
```

5.3 Obtaining the Final Solution

Now that we have solved the both the sides of the variational formulation, we have reduced the problem to a system of linear equations $AD = L$. From here, we can simply obtain D, which represents the coefficients of the linear combination of the nodal basis functions.

Hence, we obtain our approximation.

$$\tilde{u} = \Sigma D_i \phi_i$$

In the next chapter, we will take a specific example of the Poisson Equation and compare our approximation to the actual solution.

Chapter 6

Results

Now that we have made the complete solver, it is time to test it on an example. Let us consider the following problem

$$\begin{aligned} -\Delta u &= f \ in \ \Omega \\ u &= 0 \ on \ \delta\Omega \end{aligned}$$

Here, let Ω denote the cubical domain $[-1,1] \times [-1,1] \times [-1,1]$ and let $f(x,y,z) = 3\pi^2 sin(\pi x)sin(\pi y)sin(\pi z)$. It can be verified that $u(x,y,z) = sin(\pi x)sin(\pi y)sin(\pi z)$. Our goal is to approximate the function u.

6.1 Graphs and Error

Since $u : \mathbb{R}^3 \to \mathbb{R}$, to make a graph of the approximation and the actual function we take a slice of domain for a fixed value of z. The graphs of the approximation for varying step sizes have been attached. The actual function has also been graphed for comparison.

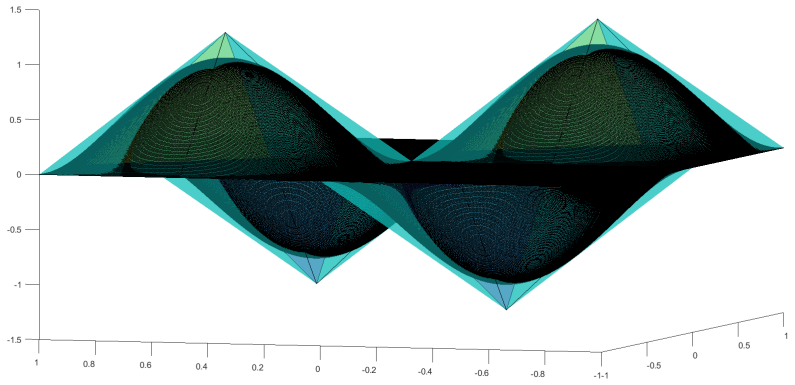

Figure 6.1: Approximation Obtained For Step Size = 1/2

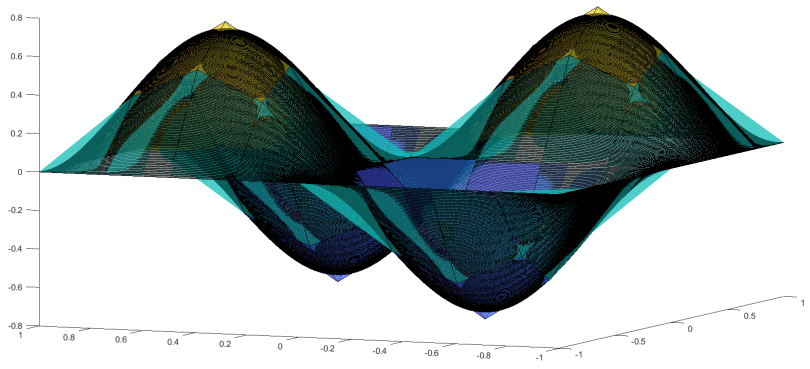

Figure 6.2: Approximation Obtained For Step Size = 1/4

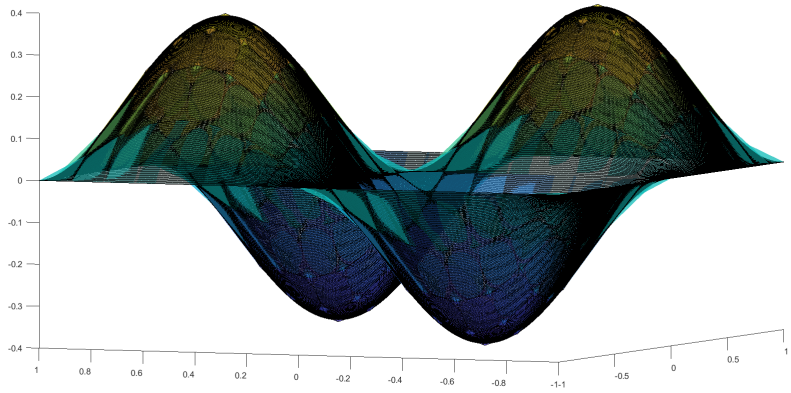

Figure 6.3: Approximation Obtained For Step Size = 1/8

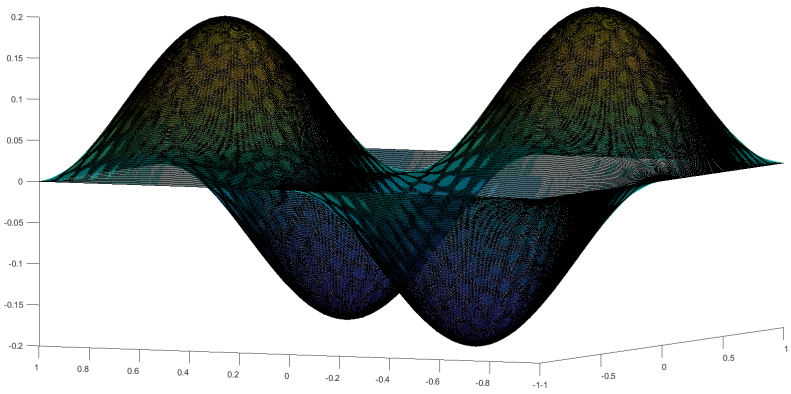

Figure 6.4: Approximation Obtained For Step Size = 1/16

It can be seen that with decreasing step sizes our approximation of the target function improves. In Figure 6.4 it is difficult to tell apart the approximation and the actual function. Let us now look at the error in approximation for different step sizes.

Since we have the value of the coefficients for the nodal basis functions, if we take the value of $\tilde{u}$ at some node, it will be equal to the coefficient of the corresponding basis function. We will calculate the error at the nodes. Let D represent the value of $\tilde{u}$ at the nodes and val represent the value of u at the nodes. The error of approximation has been taken as:

$$E = \sqrt{(D - val)^T A (D - val)}$$

For varying step sizes we get the error as:

Step Size	**Error**	E_i/E_{i-1}
0.5	1.1449	-
0.25	0.2812	0.2456
0.125	0.0700	0.2489
0.625	0.0175	0.25

Table 6.1: Step Size vs Approximation Error

The data matches our expectation that the error should be of order $O(h^2)$. Hence, with decreasing step size our error reduces and our approximation converges to the actual function.

Bibliography

[1] B. Daya Reddy. *Introductory Functional Analysis*. Springer Science+Business Media, LLC), 1998.

[2] FJ Sayas. *A Gentle Introduction to the Finite Element Method*. 2008.

[3] Eric A. Machorro Sidney Shields, Jichun Li. Weak galerkin methods for time-dependent maxwell's equations. 2017.

[4] Dan Li Yunqing Huang, Jichun Li. Developing weak galerkin finite element methods for the wave equation: Weak galerkin finite element methods. 2017.