Virtual Element Methods

MTH598 Numerical Partial Differential Equations
Project Report

Aditya Chetan
aditya16217@iiitd.ac.in

Instructor: Dr. Kaushik Kalyanaraman
October 6, 2020
Abstract

The topic of my final project in this course was “Virtual Element Methods for solving Partial Differential Equations”. My objective was to give an overview of the Virtual Element methods that have become popular for solving Partial Differential Equations. This report is divided into two parts – (i) The first part discusses a theoretical overview of the method, how the problem is structured, and how it is reduced to solving a linear system. We will not be going into depth of the content, hence, no proofs will be discussed for the results. (ii) The second part discusses some minor details about the implementation of the method in Python and the results obtained from the code for solving a sample problem on some test meshes. This report is heavily influenced from [1] which had been provided as the primary reference for this project by the instructor.
Chapter 1

Theoretical Overview

1.1 Model Problem

We will be taking our model problem as the 2D Poisson equation.

Let \( \Omega \) be a polygonal domain and consider:

\[
\begin{align*}
-\Delta u &= f \text{ in } \Omega \\
 u &= g \text{ on } \partial \Omega
\end{align*}
\] (1.1)

Here, \( f \in L^2(\Omega) \). As discussed in class during FEMs, we want to find a solution to the variational/weak form of this problem. We want to find a \( u \in H^1_0(\Omega) \) such that,

\[
a(u, v) := (\nabla u, \nabla v) = (f, v) \forall v \in H^1_0(\Omega)
\] (1.2)

Here, \((.,.)\) denotes the \( L^2(\Omega) \) inner product. By the Lax-Milgram lemma that we proved in class, the solution of this problem is guaranteed to exist and be unique.

1.2 Virtual Element Method

We partition our domain into a family of non-overlapping polygons, \( T_h \). Each of these polygonal elements shall form our local element space. Let \( h \) be the maximum diameter of the polygonal elements. Let \( E \) be any polygonal element in our set of partitions. Then, \( N_E \) is the number of edges of \( E \). Additionally, \( e_i \) is the edge between vertices \( \nu_i, \nu_{i+1} \), \( i = 1, 2, \ldots, n \). Here it is understood that the indices will wrap to 1 after \( N_E \)

Remarks:

- Notice, that we have placed no restriction on the shape of the mesh! This means that concave meshes, hanging node, everything is allowed!
- Some works, like [2], have shown that the meshes may also have arbitrary small edges.

1.2.1 Virtual Element Function Spaces

We define a discrete function space as follows:

\[
V_h := \{ v_h \in H^1_0(\Omega) : v_h \in V_h^E \ \forall E \in T_h \}
\]
Every function in $V^E_h$ is a linear polynomial on each edge of $E$.

Remarks:

- On triangular meshes, the local space consists entirely of linear polynomials. (Three vertices of each triangle can fully determine any linear polynomial).
- On more general polygonal meshes, we have higher order functions, but we need not determine them. We can simply use their values at the vertices.

### 1.2.2 Ritz Projection

The space $V^E_h$ of polynomials has properties that allow us to compute the Ritz projection, $\Pi^E : V^E_h \to P_E$ of any function in the local virtual element space. This projection takes any function in the local virtual element space onto the subspace of linear polynomials, $P_E$.

#### Properties of $\Pi^E$

1. $(\nabla (\Pi^E v_h - v_h), \nabla p)_{(0,E)} = 0 \forall p \in P_E$
2. $\Pi^E v_h = v_h$

Here, $w_h = \frac{1}{N_E} \sum_{i=1}^{n_E} w_h(v_i)$

From 1., divergence theorem and using the fact that the laplacian of a linear function (in this case, $P_E$ is zero, we get the relation,

$$(\nabla \Pi^E v_h, \nabla p) = (\nabla v_h, \nabla p) = \sum_{e \in \partial E} \int_{e} v_h n_e \nabla p ds$$

The extreme right term can be computed exactly, since $v_h$ is a linear polynomial on the edges of the local element $E$ and can therefore be completely determined by its values on the end-points of the edges. And $\nabla p$ is a given constant. Here, $n_e$ represents the outward normal from the edge $e$ of $E$.

### 1.2.3 Discrete Bilinear Form

In this section, we will follow in the footsteps of [3] to define a discrete version of our original bilinear form, $a$.

Now, since we have the discrete space set up for us, we need to define a discrete version of the original bilinear form, $A$.

We first define the local bilinear form ($a$ restricted to $E$), $a^E$, which is the bilinear form over all functions in $H^1(E)$.

Now, we define a discrete version of $a^E$ as follows:

$$a^E_h(v_h,w_h) := (\nabla \Pi^E v_h, \nabla \Pi^E w_h)_{0,E} + S^E(v_h - \Pi^E v_h, w_h - \Pi^E w_h)$$

(1.3)

In the above equation, the function $S^E(\ldots)$ is defined as:

$$S^E(v_h,w_h) := \sum_{r=1}^{N^E} \text{dof}_r(v_h) \text{dof}_r(w_h)$$

Essentially, this is the dot product of the vectors, that contain the degrees of freedoms of $v_h$, $w_h$ on the vertices of $E$ taken in a particular order.

Notice that $a^E_h$ is the local discrete bilinear form. Now, we define the global discrete bilinear form, $a_h$ as:
\[ a_h(v_h, w_h) := \sum_{E \in T_h} a^E(v_h, w_h) \]

According to [3], this discrete bilinear form must satisfy the following two properties:

- **Polynomial Consistency**: It means that for any polynomial \( p \in P_E \), and for any function \( v_h \in V_h \), we must have:
  \[ a^E(v_h, p) = a^E(v_h, p) \]
  Thus, we must have a discrete bilinear form, such that it is equal to the analytical bilinear form for any polynomial in \( P_E \)

- **Stability**: There exists a constant, \( C_{stab} \), independent of \( h \) and \( E \), such that,
  \[ C_{stab}^{-1} a^E(v_h, v_h) \leq a^E(v_h, v_h) \leq C_{stab} a^E(v_h, v_h) \]
  for any \( v_h \in V^E_h \)
  This is somewhat similar to equivalence of norms. As proved in [3], this property also implied that the discrete bilinear form is continuous and coercive.

Due to these two properties, the two terms of the discrete bilinear forms have special names:

- \( \left( \nabla \Pi^E v_h, \nabla \Pi^E w_h \right)_{0,E} \) is referred to as the consistency term as it is the only one non-zeros when one of \( v_h \) or \( w_h \) is a polynomial. (Since then the Ritz projection of that function is same as the function itself) and ensures that the consistency is satisfied.

- \( S^E (v_h - \Pi^E v_h, w_h - \Pi^E w_h) \) is called the stabilising term, as it is non-zeros even when \( v_h \) or \( w_h \) lies in the kernel of \( \Pi^E \) and ensures that stability still holds.

Now that we have the discrete formulation of the left hand side of 1.2, we can formulate the discrete counterpart of the right hand side of 1.2.

[3] shows that this can be done by using the mapping, \( \ell_h : V_h^E \rightarrow \mathbb{R} \), such that,

\[ \ell_h(v_h) := \sum_{E \in T_h} (\Pi_0^E f, \Pi_0^E v_h)_{0,E} \]  \hspace{1cm} (1.4)

Here, \( \Pi_0^E : V_h^E \rightarrow \mathbb{R} \) is a projection of functions in \( V_h^E \) onto constants, such that,

\[ \int_E (w_h - \Pi_0^E w_h) \, dx = 0 \]

Hence, we have our final discrete version of the problem, given by:

Find a \( u_h \in V_h \), such that,

\[ a_h(u_h, v_h) = \ell_h(v_h) \]  \hspace{1cm} (1.5)

for any \( v_h \in V_h \)
Chapter 2

Implementation Overview

2.1 Setting up the linear system

Let \( \{ \varphi_i \}_{i=1}^N \) be the Lagrange basis of \( V_h \), such that, \( \varphi(\nu_j) = \delta_{ij} \), where \( \delta_{ij} \) is the Kronecker Delta.

Similarly, this can be extended to the local element space \( E \), with \( \{ \varphi_i \}_{i=1}^{N_E} \) as the local Lagrangian basis. Additionally, we also need a basis for \( P_E \). Since, I was following [1] for my implementation, I also used the scaled monomials of degree 1. On the element \( E \), this basis looks like this:

\[
M_E := \{ m_1(x,y) := 1, \ m_2(x,y) := \frac{x - x_E}{h_E}, \ m_3(x,y) := \frac{y - y_E}{h_E} \}
\] (2.1)

Here, \((x_E, y_E)\) are the centroids of the element \( E \) and \( h_E \) is the diameter of the element \( E \).

In my implementation, I calculate \( x_E \) and \( y_E \) using the expression given in [1], which are:

\[
x_E = \frac{1}{6|E|} \sum_{i=1}^{N_E} (x_i + x_{i+1})(x_{i+1}y_i - x_iy_{i+1})
\]

\[
y_E = \frac{1}{6|E|} \sum_{i=1}^{N_E} (y_i + y_{i+1})(x_iy_{i+1} - x_{i+1}y_i)
\] (2.2)

The indices are wrapped around \( N_E \). Here \(|E|\) is the area of element \( E \) and is further given by:

\[
|E| = \frac{1}{2} \sum_{i=1}^{N_E} x_iy_{i+1} - x_{i+1}y_i
\] (2.3)

Now, first, we express \( u_h \) as the linear combination of the basis of \( V_h \) as,

\[
u_h = \sum_{i=1}^{N} U_i \varphi_i
\]

Now, we use, this expansion, along with 1.3 and 1.4 can be used to re-write 1.5 as:

\[
\sum_{i=1}^{N} U_i \sum_{E \in T_h} \left( (\nabla \Pi^E \varphi_i, \nabla \Pi^E \varphi_j)_{0,E} + S^E \left( \varphi_i - \Pi^E \varphi_i, \varphi_j - \Pi^E \varphi_j \right) \right) = \sum_{E} (\Pi^E_0 f, \varphi_j)_{0,E} \] (2.4)

for every \( j = 1, 2, \ldots, N \). This can be expressed as the linear system, \( KU = F \), where,

\[
K_{j,i} = \sum_{E \in T_h} \left( (\nabla \Pi^E \varphi_i, \nabla \Pi^E \varphi_j)_{0,E} + S^E \left( \varphi_i - \Pi^E \varphi_i, \varphi_j - \Pi^E \varphi_j \right) \right), \quad F_j = \sum_{E} (\Pi^E_0 f, \varphi_j)_{0,E}
\] (2.5)

for \( i, j = 1, 2, \ldots N \).
Similarities and Differences from FEM

Similarities

- Since we now have a linear system, the overall structure of the method will be more or less the same as Finite Element methods, i.e., assembling $K$ using local element stiffness matrices along with $F$ and then solving the linear system, taking into account the boundary conditions.

Differences

- Here, the way the local element stiffness matrix $K_E$ is assembled would be different, as it would require the calculation of the local Ritz projection, $\Pi^E$ on each element.

- Additionally, in FEM, we have a reference element, that gives us a shape function, that tells us how the local basis in each element would look like. Here, we do not have any mapping to any reference element. This is because, VEM allows non-convex polygons, and according to [1], such mappings are not possible in the case of general polygons.

Ritz Projection, Local Stiffness Matrix and Forcing Vector

For the implementation of this part, I referred to [1]. I will not be discussing this portion again in this report. Readers may refer to [1] for a detailed understanding.

2.2 Analysis & Results

I implemented a Python version of the MATLAB code provided by the authors of [1]. I tried to replicate the results that the authors had shared in the paper. My analysis involved:

- Recreating the plots on the meshes and boundary conditions shared by the authors as a part of their work in [1].

- Checking the accuracy of my code by comparing it with the results generated by the code provided by [1]. I took the output of their code as the true value and calculated the $L_2$ norm error in my solution.

2.2.1 Results

I performed analysis on Square-shaped domain and L-shaped domain, following [1].

Square-shaped Domain

On this domain, my RHS was:

$$f = 15 \sin(\pi x) \sin(\pi y)$$

and my boundary condition was:

$$u = (1 - x)y \sin(\pi x) \text{ on } \partial \Omega$$

My plots for this domain on different meshes are given in Figure 2.1 and error analysis is shown in Table 2.1.
(a) Square mesh.  
(b) Triangle Mesh.  
(c) Voronoi Mesh.  
(d) Smoothened Voronoi Mesh.  
(e) Non-convex Mesh.

Figure 2.1: Results on a Square Domain.

<table>
<thead>
<tr>
<th>Mesh Type</th>
<th>Absolute Error</th>
<th>Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>$4.35 \times 10^{-5}$</td>
<td>$5.68 \times 10^{-6}$</td>
</tr>
<tr>
<td>Triangle</td>
<td>$3.85 \times 10^{-5}$</td>
<td>$5.76 \times 10^{-6}$</td>
</tr>
<tr>
<td>Voronoi</td>
<td>$7.41 \times 10^{-5}$</td>
<td>$5.60 \times 10^{-6}$</td>
</tr>
<tr>
<td>Smoothened Voronoi</td>
<td>$7.65 \times 10^{-5}$</td>
<td>$5.65 \times 10^{-6}$</td>
</tr>
<tr>
<td>Non-Convex</td>
<td>$9.28 \times 10^{-5}$</td>
<td>$5.61 \times 10^{-6}$</td>
</tr>
</tbody>
</table>

Table 2.1: Error Analysis for Square-shaped domain
L-shaped domain

On this domain, my RHS was:

\[ f = 0 \]

and my boundary condition was:

\[ g(r, \theta) = r^{2/3} \sin \left( \frac{2\theta - \pi}{3} \right) \]

where \( r \) and \( \theta \) are the standard polar coordinates of the point from the origin.

My plot for this domain is given in Figure 2.2. Due to some interpolation issues in the matplotlib library that I used for plotting, the resulting plot also shows values outside of the domain. As of the time of writing I am still working on a solution to this problem.

Error Analysis:

- Absolute Error: \( 1.38 \times 10^{-4} \)
- Relative Error: \( 1.46 \times 10^{-5} \)
Bibliography

