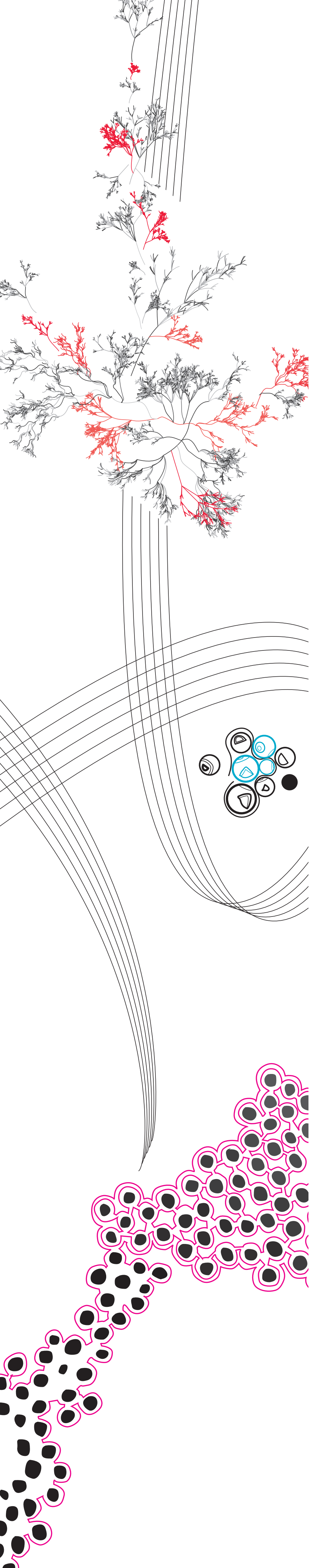


April 13, 2018

MASTER THESIS

A comparison of polygonal discontinuous Galerkin methods

L.J. Corbijn van Willenswaard

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Chair: Mathematics of Computational Sciences (MACS)

Assesment committee:
Prof. dr. ir. J.J.W. van der Vegt
S. Geevers MSc
Dr. M. Schlottbom

UNIVERSITY OF TWENTE.

Acknowledgements

This thesis is the result of not only my master project, but also of almost nine years of learning and working. At this finishing point, I am looking back at an interesting and challenging final project and good and educational time as student. Such time would not have been possible with the support and guidance of several people, to whom I would like to express my gratitude.

First of all, to the members of my assessment committee. Specifically, to Jaap van der Vegt for giving me this project and his guidance. I have enjoyed working on it, so much that I sometimes needed some persuasion to do more writing and less experimenting. To Sjoerd Geevers, for providing me feedback when I needed it and the first correction of much of this thesis. Finally, to the external member Matthias Schlottbom, for joining the committee and for providing me with some small writing tips.

Secondly, I have to thank my internship supervisor Sander Rhebergen. In my internship I learned about discontinuous Galerkin methods and implementing finite element codes. It safe to say that without it, I would not have progressed so far in this project.

I also have to thank my fellow office members, Nishant, Olena, Poorvi, Sjoerd and Sjoerd. Giving me some nice discussions and good time in the office.

Looking back at my nearly nine years in Twente, I need to thank my friends and colleagues for wonderful times. Specifically, I want to thank Caroline, for all the discussion, fun and support over the last few years.

Lastly, I have to thank my parents and family. Without you, I would not have come this far in life.

Contents

Chapter 1

Introduction

The ultimate test for any numerical physics computation is the comparison with results from actual experiments. Such a comparison is a careful balancing act, we can not capture reality without a detailed model of the experiment, while the computation becomes too expensive when adding too much detail. Removing these details requires knowledge about the physics, for example in optics we can usually remove details that are much smaller than the wave length of light. But it is not always easy to judge what is small enough to be removed and what, and how this removal should be done. The motivation for this project has exactly such a problem with a photonic crystal.

A photonic crystal is a nanostructure that has spatially periodic patterns on a size comparable to the wave length of light. By careful design of this structure a photonic band gap can be created, thereby preventing the existence of light of certain frequencies in the crystal. Computations of the structure shows that in theory this results in a 100% reflectivity [27]. Reality is however different, the inevitable manufacturing defects reduce the effectiveness of the crystal.

With the recent, non-destructive, imaging methods it is possible to obtain the structure of actual crystals at 20 nm resolution [36]. Using this data as basis for a numerical computation would allow us to compute the properties of an actual crystal and compare them with experimental reality.

The problem with this approach is the level of detail in the crystal. The imaged piece of the crystal is $(20\ \mu\text{m})^3$ in size, resulting in 10^9 voxels with a value of the electron density. This density can be used to reconstruct the hundreds of tubes etched into the silicon to create the crystal structure. A traditional finite element mesh created from this data would have many very small elements. Which is caused by the non-smooth silicon-air interface boundary of each of the tubes, for which we have details that are of a size comparable to the imaging accuracy of 20 nm.

Abstracting the actual problem, we see that the problem is the very detailed interface, which requires equally many small elements. So many that we expect that tetrahedral meshes in combination with traditional finite element approaches become computationally infeasible.

Such problems with too detailed interfaces are not limited to applications in nano-photonics. ADG has, for example, already been applied to computationally estimating the effective Young's modulus for a piece of trabecular bone [22]. While VEM has been tested for complicated geomechanical problems, where almost every element has other material properties [4].

1.1 Research question and methods

In this thesis, we look at finite element methods that allow more general polygonal and polyhedral elements. With the idea, that the more flexible element shapes would reduce the effect of the geometric complexity on the mesh size [10]. Thereby, allowing a reduction of the number of elements to a computationally feasible amount.

Our main question is, therefore, what is the best finite element method available to solve problems where the geometric complexity of an interface creates too many small elements. Specifically, if possible we want a Discontinuous Galerkin method, as these are currently used to solve the Maxwell equations for nano-photonic applications [37].

As the original problem is far beyond the scope of a Master Thesis, we will use a generalized Poisson equation in 2D as model problem. Extending the standard finite element approach to polygonal elements, to solve our model problem, does result in some complications. The most important one is that conforming basis functions can not be polynomial [53]. Thus, methods have to choose between using polynomial, but non-conforming basis functions or conforming, but non-polynomial basis functions. The two methods that we selected from literature to study in this thesis make a fundamentally different choice.

The Agglomerated Discontinuous Galerkin method (ADG) [10] chooses to use polynomial basis functions. Where the actual basis functions are constructed by orthogonalizing a set of monomials. That these basis functions are not conforming is no problem for the DG approach that we want to take, while the polynomial property allows an easier implementation and analysis.

The Virtual Element Method (VEM) [25] and its DG counterpart DGVEM [16], choose to use conforming basis functions, which require several approximations and projections to not compute them inside the element. These virtual basis functions come as a double-edged sword. They allow customization of the element space to, for example, allow arbitrary levels of continuity, or be $H(\text{curl})$ conforming. On the other hand, the requirement of not computing the values of the basis functions inside the element makes the method more complex than standard finite element methods.

We show, with numerical test, what the implications are of this different choice between polynomial and conforming basis functions. We can clearly observe that ADG has a higher order accuracy, but that this is less useful when the elements are relatively large. We show several examples of test cases where a DGVEM solution has a lower error compared to the ADG solution with the same number of degrees of freedom, even though that used DGVEM has a lower order of polynomial accuracy.

1.2 Outline

The outline of this thesis is as follows.

In Chapter 2 we start by defining our model problem and introduce the essential notation. Additionally we also introduce the SIPG method [28] in physical space, which serves as reference method.

This description of the reference method is used in Chapter 3 to discuss the problems with using polygonal and polyhedral elements and the solutions available in literature. Motivating why we selected ADG and DGVEM as the most prospective candidates.

In Chapters 4 and 5, we will describe these methods. Furthermore, as there are no standard quadrature rules on polygonal or polyhedral elements, we will look at alternative approaches for computing the necessary integrals.

Both of these methods are put to test in several numerical tests in Chapter 6. We compare the alternatives for the element vector and norm computation to determine their computational cost and accuracy. Additionally, we compare the results of ADG and DGVEM for several test cases that represent our intended application.

Finally, in Chapter 7, we will summarize the results and present our recommendations. In addition, we look back at the report and present some additional topics for future research.

Chapter 2

Model problem and physical space SIPG

Before going into the alternative methods, we will first look at a more traditional discontinuous Galerkin approach in the form of SIPG. We start by defining the model problem that we will use to explain and test the methods in this report. After introducing some necessary notation, we define the traditional SIPG method and a physical space version of it. We finish with a small description of the verification used to test the methods.

2.1 Model problem

We will use a generalized Poisson problem as model problem. Specifically, let $\Omega \subset \mathbb{R}^d$ be a d-dimensional open polytopic domain of computation, where the boundary $\partial\Omega$ can be decomposed into two independent parts Γ_D and Γ_N, with Γ_D having postive measure. Then our problem is to find $u \in H^1(\Omega)$ such that

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma_D, \\ \mathbf{n} \cdot \nabla u = 0 & \text{on } \Gamma_N. \end{cases} \tag{2.1}$$

Here $f \in L^2(\Omega)$ is the source term, g_D is a sufficiently regular Dirichlet boundary condition and $\mathbf{n}$ is the outward pointing normal. The interpretation of the function κ depends on the application of the problem, with possibilities such as diffusivity for the diffusion equation and permeability for Darcy's Law. The general conditions on κ vary. We will restrict it to a piecewise constant. But in more general setting it can be a (positive definite) tensor, thereby introducing some anisotropy into the problem.

While the formulation in (2.1) is convenient for interpretation, it is not convenient for finite element methods. Therefore, we will instead solve the weak formulation of (2.1), which is finding $u \in H^1(\Omega, g_D)$ such that

$$\int_\Omega \kappa \nabla u \cdot \nabla v \,\mathrm{d}\mathbf{x} = \int_\Omega f v \,\mathrm{d}\mathbf{x} \quad \forall v \in H^1(\Omega, 0), \tag{2.2}$$

where

$$H^1(\Omega, g) = \left\{u \in H^1(\Omega) \ : \ u|_{\Gamma_D} = g\right\}.$$

are the functions in the standard Sobolev space $H^1(\Omega)$ with trace g_D.

2.2 Notation

Before going into the details of the SIPG method used for solving the problem, we will introduce some notation.

We create a mesh $\mathcal{T}_h$ of the domain Ω with d-dimensional polytopic elements. This mesh should precisely cover the domain ($\cup_{E\in\mathcal{T}_h}\bar{E} = \bar{\Omega}$) and the elements, that we assume are open, do not overlap. Moreover, we assume that κ, which is piecewise constant on Ω, is constant in each element of $\mathcal{T}_h$. As usual for DG methods, we will define $\mathcal{E}_h$ as the edges of our mesh, which can be split into those on the Dirichlet boundary $\mathcal{E}_h^{\partial,D}$, Neumann Boundary $\mathcal{E}_h^{\partial,E}$ and the internal edges $\mathcal{E}_h^0$. For convenience we also write $\mathcal{E}_h^{\partial} = \mathcal{E}_h^{\partial,D} \cup \mathcal{E}_h^{\partial,E}$ and $\mathcal{E}_h^{0,D} = \mathcal{E}_h^{\partial,D} \cup \mathcal{E}_h^0$.

We will use the letters E and e as generic letters to refer to an element (E) and an edge (e) on this mesh. The measure of them is denoted by $|E|$ and $|e|$ and their diameter by h_E and h_e. Note that for edges we have $|e| = h_e$. For an element E, we will denote its centroid by $\mathbf{x}_E$ and its number of vertices by $n_{E,v}$.

For any domain D, we denote the space of polynomials of degree at most k by $\mathcal{P}_k(D)$, where, by convention, we set $\mathcal{P}_{-1}(D) = 0$. Both of the methods we will select use the scaled monomials as basis for this space. When considering a two-dimensional problem, let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2) \in \mathbb{N}^2$ be a multi-index and let $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ be any vector. We define $|\boldsymbol{\alpha}| = \alpha_1 + \alpha_2$ and $\mathbf{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} x_2^{\alpha_2}$. Extension to a higher dimension is trivial by adding the extra terms $|\alpha|_i$ and $x_i^{\alpha_i}$ for each of the extra dimensions. Using this notation, we define the set of scaled monomials of degree at most k on an element E as

$$\mathcal{M}_k(E) = \left\{ \left(\frac{\mathbf{x} - \mathbf{x}_E}{h_E}\right)^{\boldsymbol{\alpha}} \ : \ |\boldsymbol{\alpha}| \le k \right\}.$$

The scaled monomials of precisely order k, are given by

$$\mathcal{M}_k^{\star}(E) = \left\{ \left(\frac{\mathbf{x} - \mathbf{x}_E}{h_E}\right)^{\boldsymbol{\alpha}} \ : \ |\boldsymbol{\alpha}| = k \right\}.$$

For more practical matters (such as implementing them) and for the convenience of not needing a vector subscript, we will order the scaled monomials. We associate each multi-index and the monomial generated by it with an element of $\mathbb{N} \setminus \{0\}$[1]. For this, we use a slightly modified version of the Cantor pairing function:

$$\pi(\boldsymbol{\alpha}) = \frac{1}{2}\,|\boldsymbol{\alpha}|\,(|\boldsymbol{\alpha}| + 1) + \alpha_2 + 1, \tag{2.3}$$

which generates the sequence

$$1 \leftrightarrow (0,0), 2 \leftrightarrow (1,0), 3 \leftrightarrow (0,1), 4 \leftrightarrow (2,0), \ldots. \tag{2.4}$$

[1]The only reason for excluding zero is that the matrix and vector indices in MATLAB start at 1.

Using this function, we associate with each multi-index a number and write $\boldsymbol{\alpha}_i$ for the multiindex with $\pi(\boldsymbol{\alpha}) = i$. Similarly, we write m_i for the scaled monomial with $\boldsymbol{\alpha}_i$ as power.

Having done all this notational work, we can now expand a polynomial $p \in \mathcal{P}_k(E)$ in terms of its monomial coefficients $\mathbf{p}_i$

$$p(\mathbf{x}) = \sum_{i=1}^{n_{\mathcal{P},k}} \mathbf{p}_i m_i(\mathbf{x}),$$

which allows us to represent a polynomial p as a vector $\mathbf{p}$ of coefficients. This ordering is not really interesting from a mathematical point of view, but it is needed for the implementation and for simplifying the notation.

To present the discontinuous Galerkin methods, it is convenient to introduce the jump and average of functions that are double valued on edges of the mesh. Let g and $\mathbf{g}$ be such functions that are respectively scalar and vector valued. Let E^+ and E^- be two different elements in $\mathcal{T}_h$ such that they share an edge $e = \partial E^+ \cap \partial E^-$ and let $\mathbf{n}^\pm$ be the outward pointing normal of e with respect to element $E^\pm$. Let $g^\pm$ be the trace of g on e with respect to $E^\pm$ and analogously define $\mathbf{g}^\pm$. Then the jump $[\![\cdot]\!]$ and weighted the average $\{\!\{\cdot\}\!\}_{\delta_e}$ with weight $\delta_e \in [0,1]$ are defined as

$$\{\!\{g\}\!\}_{\delta_e} = \delta_e g^+ + (1-\delta_e)g^-, \qquad [\![g]\!] = g^+\mathbf{n}^+ + g^-\mathbf{n}^-, \tag{2.5}$$

$$\{\!\{\mathbf{g}\}\!\}_{\delta_e} = \delta_e \mathbf{g}^+ + (1-\delta_e)\mathbf{g}^-, \qquad [\![\mathbf{g}]\!] = \mathbf{g}^+\cdot\mathbf{n}^+ + \mathbf{g}^-\cdot\mathbf{n}^-. \tag{2.6}$$

To simplify notation we also define these functions on a boundary edge $e \in \mathcal{E}_h^\partial$:

$$\{\!\{g\}\!\}_{\delta_e} = g, \quad [\![g]\!] = g\mathbf{n}, \quad \{\!\{\mathbf{g}\}\!\}_{\delta_e} = \mathbf{g}, \quad [\![\mathbf{g}]\!] = \mathbf{g}\cdot\mathbf{n}. \tag{2.7}$$

For the regular average with $\delta_e = \frac{1}{2}$ we drop the subscript. In addition to these averages we will need the edge weights from [29],

$$\beta_e = \frac{\kappa^-}{\kappa^+ + \kappa^-},$$
$$\kappa_e = \frac{2\kappa^+\kappa^-}{\kappa^+ + \kappa^-}.$$

With $\kappa^\pm$ the values of κ on the element $E^\pm$. These share the relation

$$\{\!\{\kappa g\}\!\}_{\beta_e} = \kappa_e \{\!\{\cdot\}\!\}_g$$

To integrate all the functions we will use numerical quadrature rules. The standard quadrature rule on a domain D accurate for polynomials up to order l is denoted by $\mathcal{Q}_l(D)$. It is used to integrate $f \in \mathcal{P}_l(D)$ as follows

$$\int_D f \,\mathrm{d}\mathbf{x} = \sum_{(\mathbf{x}_q, w_q)\in\mathcal{Q}_l(D)} f w_q\big|_{\mathbf{x}=\mathbf{x}_q}.$$

Where we used $f w_q|_{\mathbf{x}=\mathbf{x}_q}$ instead of $f(\mathbf{x}_q)w_q$ to preserve space, as the integrands will usually be more complex than a single function. In addition to this generic quadrature rule, we will use $\mathcal{Q}_l^{lob}(e)$ and $\mathcal{Q}_l^{leg}(e)$ for the Gauss-Lobatto and Gauss-Legendre rules for an edge e. When using n, points they are accurate for 1D polynomials up to order $2n-3$ and $2n-1$, respectively.

Finally, we introduce the common notation for the inner product and norm. Specifically, we denote the standard L^2-inner product by $(\cdot,\cdot)_D$ or $\langle\cdot,\cdot\rangle_D$ when D is a d-dimensional or $d-1$-dimensional domain, respectively. Moreover, we denote the L^2-norm over a domain D by $\|\cdot\|_D$.

2.3 SIPG

With all notation introduced we can define the SIPG method to solve (2.2). It approximates the solution u by a function u_h in the space

$$V_k(\mathcal{T}_h) = \left\{g \in L^2(\mathcal{T}_h) \ : \ \forall_{E\in\mathcal{T}_h} \ g|_E \in \mathcal{P}_k(E)\right\},$$

with $g|_E$ the restriction of g to the element E. Note that, as usual in discontinuous Galerkin methods, these functions can be discontinuous at the element boundary. As a consequence, we need to add several terms to the weak formulation [8]. One of the resulting methods is SIPG, which is formulated as

$$\text{Find } u_h \in V_k(\mathcal{T}_h) \text{ such that } \forall_{v_h\in V_k(\mathcal{T}_h)} \ \mathcal{A}(v_h, u_h) = \mathcal{F}(v_h), \tag{2.8}$$

where $\mathcal{A}$ and $\mathcal{F}$ are the bilinear and linear forms given by

$$\mathcal{A}(v_h, u_h) = \sum_{E\in\mathcal{T}_h} \int_E \nabla v_h \cdot \kappa \nabla u_h \,\mathrm{d}\mathbf{x} + \sum_{e\in\mathcal{E}_h} \int_e -\{\!\{\kappa\nabla v_h\}\!\}_{\beta_e} [\![u_h]\!] - \{\!\{\kappa\nabla u_h\}\!\}_{\beta_e} [\![v_h]\!] + \gamma_e [\![v_h]\!][\![u_h]\!] \,\mathrm{d}S, \tag{2.9}$$

$$\mathcal{F}(v_h) = \sum_{E\in\mathcal{T}_h} \int_E v_h f \,\mathrm{d}\mathbf{x} + \sum_{e\in\mathcal{E}_h^{\partial,D}} \int_e g_D \left[-\mathbf{n}\cdot\kappa\nabla v_h + \gamma_e v_h\right] \mathrm{d}S. \tag{2.10}$$

To compensate for the possible discontinuities in κ between elements we used the weighted averaging from [29]. For the stabilization parameter γ_e we use

$$\gamma_e = \frac{\gamma\kappa_e}{h_e},$$

with $\gamma > 0$ a sufficiently large number.

For implementation we proceed as usual: we assume we have a basis $\{\phi_i\}_{i=1}^{N_{dof}}$ for $V_k(\mathcal{T}_h)$, with N_{dof} the total number of basis functions. Using this, we expand u_h in terms of the basis functions:

$$u_h = \sum_{j=1}^{N_{dof}} \hat{u}_j \phi_j.$$

Additionally, set $v_h = \phi_i$, for $1 \le i \le N_{dof}$, and plug this into (2.8). After some rewriting, the resulting system can be written as a large linear system of the form

$$A\mathbf{u} = F,$$

where $\mathbf{u}$ is a vector with the coefficients $\hat{u}_i$, $A_{ij} = \mathcal{A}(\phi_i, \phi_j)$ and $F_i = \mathcal{F}(\phi_i)$. Using basis functions with support on only a single element we see that most entries of the matrix A will be zero. To further simplify A and F, we split (2.9) and (2.10) into a summation of integrals over elements and edges:

$$A = \sum_{E\in\mathcal{T}_h} A^E + \sum_{e\in\mathcal{E}_h} A^e, \tag{2.11}$$

$$F = \sum_{E\in\mathcal{T}_h} F^E + \sum_{e\in\mathcal{E}_h^{\partial,D}} F^e, \tag{2.12}$$

with

$$A_{ij}^{E} = \kappa \int_{E} \nabla\phi_i \cdot \nabla\phi_j \,\mathrm{d}\mathbf{x}, \tag{2.13}$$

$$A_{ij}^{e} = \int_{e} -\{\!\{\kappa\nabla\phi_i\}\!\}_{\beta_e} [\![\phi_j]\!] - \{\!\{\kappa\nabla\phi_j\}\!\}_{\beta_e} [\![\phi_i]\!] + \gamma_e [\![\phi_i]\!][\![\phi_j]\!] \,\mathrm{d}S, \tag{2.14}$$

$$F_{i}^{E} = \int_{E} \phi_i f \,\mathrm{d}\mathbf{x}, \tag{2.15}$$

$$F_{i}^{e} = \int_{e} g_D \left[-\mathbf{n}\cdot\kappa\nabla\phi_i + \gamma_e\phi_i\right] \mathrm{d}S. \tag{2.16}$$

Using this split, we can use the fact that most of the basis functions do not have support on E or e by only computing the entries for indices i and j when their corresponding basis functions have support on the element or edge.

2.3.1 Basis functions

So far, we have done without the exact description of the basis functions, nor did we require that our mesh $\mathcal{T}_h$ consists of simplexes. The reason for this restriction comes from having to define practical basis functions, a mapping function and having to perform numerical quadrature. Both are, as we will see from the alternatives in the next chapter, much more difficult if we do not use simplexes.

For now, we will follow the traditional path and, to make life easier, assume we work in two dimensions on a triangular mesh. Additionally, we use nodal basis functions. These basis functions are zero on all but one element. For each element E, there are $n_{\mathcal{P},k}$ basis functions ϕ_i corresponding to a set of points $\mathbf{x}_i$ on that element, such that each basis function is defined as the polynomial in $\mathcal{P}_k(E)$ that solves

$$\phi_i(\mathbf{x}_j) = \delta_{ij}, \tag{2.17}$$

with δ_{ij} the Kronecker delta function.

The standard points to use for these basis functions are evenly spaced in barycentric coordinates, thus $\frac{1}{k}(i_1, i_2, i_3)$, with $i_1 + i_2 + i_3 = k + 1$ and $i_1, i_2, i_3 \in \mathbb{N}$. These basis functions have several nice properties that we will discuss in the next chapter.

To compute the basis functions we expand them in terms of the scaled monomials:

$$\phi_j = \sum_{i=1}^{n_{\mathcal{P},k}} b_{ij} m_i.$$

For these monomials, we can easily calculate the values at the points $\mathbf{x}_j$. If we combine this with (2.17) we get a matrix system of the form

$$MB = I. \tag{2.18}$$

Where M_{ij} is the value of monomial m_j at point $\mathbf{x}_i$, B is the matrix with coefficients b_{ij} and I is the identity matrix of size $n_{\mathcal{P},k}$. Thus, the coefficient matrix B can be computed by inverting M.

The larger k gets, the more expensive this inverting of M becomes. Hence the use of a reference element in standard finite element methods. Instead of computing the basis function on the actual element, we can also compute them on a predefined reference triangle $\hat{E}$. Combining this with a simple linear mapping from $\hat{E}$ to E gives us the actual basis functions.

2.3.2 Quadrature

The use of a reference element is closely tied with another part of the implementation of finite element methods: the use of quadrature rules. These are needed to perform the element and edge integration of (2.13)–(2.16). We use the first equation to demonstrate this.

To compute (2.13) we first perform a change of basis from the physical space coordinates of element E to the coordinates for the reference element $\hat{E}$.

$$\begin{aligned} A^E_{ij} &= \int_E \kappa \nabla \phi_i \cdot \nabla \phi_j \, \mathrm{d}\mathbf{x}, && (2.19)\\ &= \int_{\hat{E}} \kappa_E \nabla \phi_i \cdot \nabla \phi_j \, |J| \, \mathrm{d}\mathbf{x}, && (2.20)\\ &= \int_{\hat{E}} \kappa_E (J^{-T} \hat{\nabla} \hat{\phi}_i) \cdot (J^{-T} \hat{\nabla} \hat{\phi}_j) \, |J| \, \mathrm{d}\mathbf{x}. && (2.21) \end{aligned}$$

Where κ_E is the value of κ on E, J is the Jacobian of the transformation $\tau_E : \hat{E} \to E$, $\hat{\phi}_i$ is the basis function on the reference element and $\hat{\nabla}$ is the gradient operator in the coordinate space of the reference element.

Evaluating this integral can become rather complicated due to the factors introduced by the Jacobian. Therefore it is calculated using a numerical quadrature rule consisting of points $\hat{\mathbf{x}}_q$ on $\hat{E}$ and corresponding weights $\hat{w}_q$. When defining the basis functions on the reference element, we use numerical quadrature for evaluating (2.21) to get

$$A^E_{ij} \approx \sum_{(\hat{\mathbf{x}}_q, \hat{w}_q) \in \mathcal{Q}_{2k}(\hat{E})} \kappa (J^{-T} \hat{\nabla} \hat{\phi}_i) \cdot (J^{-T} \hat{\nabla} \hat{\phi}_j) \, |J| \, \hat{w}_q \Big|_{\hat{\mathbf{x}} = \hat{\mathbf{x}}_q} . \qquad (2.22)$$

For the physical space coordinate system, we transform the quadrature rule from the reference element $\hat{E}$ to the physical space:

$$\mathcal{Q}_{2k}(E) = \left\{ (\tau_E(\hat{\mathbf{x}}_q), |J| \, \hat{w}_q) \ : \ (\hat{\mathbf{x}}_q, \hat{w}_q) \in \mathcal{Q}_{2k}\left(\hat{E}\right) \right\}.$$

Applying this transformed quadrature rule to evaluate (2.20), we get

$$A^E_{ij} \approx \sum_{(\mathbf{x}_q, w_q) \in \mathcal{Q}_{2k}(E)} \kappa \nabla \phi_i \cdot \nabla \phi_j w_q \big|_{\mathbf{x} = \mathbf{x}_q} , \qquad (2.23)$$

where the Jacobian $|J|$ from (2.20) is absorbed into the quadrature weights.

Note (again) the connection to the section on calculating the basis functions. In (2.22) we only need to calculate $\hat{\nabla} \hat{\phi}_i$ at the integration points of the reference element. As these points are the same for each element E we can precompute these values, thereby reducing the dominant cost of the integration to computing $|J|$ and J^{-1} for each integration point on each element E.

Whether this detour with a reference element is worth the effort depends on the case at hand. When using the reference element, we have to calculate J^{-1}, a $d \times d$ matrix, for each integration point on every element at the advantage of only calculating the basis functions and their gradients once. Sparing at least the work for inverting the $n_{\mathcal{P},k} \times n_{\mathcal{P},k}$ matrix M for every element to construct the basis functions. As the cost of such inversion grows much faster than the number of points in the quadrature rule, we see that using a reference element gets more efficient with increasing order.

This performance argument is not the only reason for using a reference element. It also allows for using elements with curved edges using isoparametric mappings. Furthermore, we not only need to compute the basis functions only once, we also only need to store their coefficients once, reducing the required memory of an implementation.

2.4 Verification

In addition to implementing the algorithm, we also need to verify that it works as expected. The standard procedure for this is the method of manufactured solutions, where we start with an analytic solution u and derive the corresponding f. With that, we compute u_h on meshes of different size and see if the error conforms to the analytical bound

$$\|u - u_h\|_\Omega \leq Ch^{p+1} \|u\|_\Omega$$

Where C is some positive constant not depending on u or the mesh size h.

The calculation of this norm is relatively straight forward:

$$\|u - u_h\|_\Omega = \sqrt{\sum_{E \in \mathcal{T}_h} \|u - u_h\|_E^2} = \sqrt{\sum_{E \in \mathcal{T}_h} \int_E (u - u_h)^2 \, \mathrm{d}\mathbf{x}}.$$

To compute the integrals we use the quadrature rule:

$$\int_E (u - u_h)^2 \, \mathrm{d}\mathbf{x} \approx \sum_q (u - u_h)^2 |J| \hat{w}_q \Big|_{\mathbf{x}=\mathbf{x}q}. \tag{2.24}$$

Note, the value $(u - u_h)^2 |J| \hat{w}_q$ will be positive if $\hat{w}_q$ is positive. Therefore, even with some rounding errors, we can be sure that the result will be positive.

2.5 Conclusion

In this chapter we introduced several key parts that we will use in our discussion of the possible alternative methods. We started with the generalized Poisson problem that will serve as model problem. We continued with some notation that will be used in the rest of this report.

We finished with a general description of SIPG for the model problem. It contains several key implementation details, such as the use of a reference element, which become expensive or impossible in the alternative methods.

Chapter 3

Selecting suitable methods

As already discussed in the introduction, we expect that the more standard finite element methods, like the SIPG method from the previous chapter, require too large meshes on very complicated domains. Hence, we need to find a different method. In this chapter, we will discuss the literature on different variations of finite element methods that might provide an alternative. This is done in a multi-step process.

We start by defining the criteria that our method should (preferably) have. Using these criteria we can quickly rule out all but three methods as unfavourable. The remaining candidate methods are then discussed in more detail. Finally, using the advantages and disadvantages from the different methods in this discussion, we will make a choice for the most suitable methods.

3.1 Criteria

As the first step in the process of finding suitable methods, we need to define what we consider a suitable method. These criteria can be divided into two sets: some essential requirements, e.g. it should solve our problem, and some nice-to-have properties, e.g. available analysis on the method.

3.1.1 Essential requirements

There is only one essential requirement for the method, which can be summarized as: it should solve our mathematical problem. Our problem is that, due to the geometric complexity of the interface or boundary, the elements locally need to be very small and numerous. So numerous that traditional finite element methods become too computationally expensive.

More specifically we assume that our domain of computation contains one or more interfaces. The shape of these interfaces can be split into two components: a general shape and a deviation from it that capture the comparatively small details. We assume that a FEM computation on a mesh that only resolves the general shape of the interface is computationally feasible, while a mesh that also resolves the fine details of the interface would introduce so many extra elements that traditional finite element methods would become computationally infeasible.

Thus, the main requirement is that the method is able to resolve the small details in the interface in a computationally feasible manner. This can be done by, for example not using a mesh, or by using a mesh with different shapes that allow for larger elements.
Remark. *This implicitly assumes that the details on the mesh are not so small that handling them is computationally infeasible in the first place. What the minimal size is of the details before a method becomes computational infeasible is of course dependant on the method in question.*

3.1.2 Useful properties

In addition to the requirement that the method should solve our problem, we would prefer the method to have several other characteristics. We have the following preferences:

- We prefer methods that can be converted to some type of DG approach, since this type of method is currently used by the research group. Note though, that most conforming basis functions can be adapted to be used in DG methods.
- To be more useful for future use, beyond the application from the introduction, we would like a method that allows higher order basis functions.
- For convenience, we prefer methods for which there is some analysis available.
- We would like methods that are simple, both in description and implementation.
- To reduce the additional cost incurred by using the alternative method, it would be nice if we can couple it to more traditional methods. This would allow a mesh where the alternative method is only used on the subset of the elements where it is needed.

3.2 Disregarded methods

There were several methods that sound interesting, but, after a some study, were dropped as possible candidates. For the sake of completeness and background we give a short summary, including the reason why we do not expect them to solve our problem.

3.2.1 Meshless methods

Meshing, even without complicated microstructures, is a hard problem. The presence of interfaces adds the requirement that the mesh should resolve the details of these structures while having a structure that does not result in large computational errors. Thus an obvious choice would be to go meshless.

There are many versions of meshless algorithms, see, for example, [45], some more related to FEM than others. The main idea is that they decompose the solution into shape functions around nodes placed on the domain, but, unlike FEM, the nodes are not connected to a mesh, nor is the solution necessarily an interpolation where the value at the node is only determined by the nodal value.

These strengths also give rise to the weaknesses. The incorporation of Dirichlet boundary conditions is quite a bit harder with meshless methods. More of a problem are the internal interfaces. With a mesh, we can approximate the interface and separate the shape functions on both sides of the interface. For meshless methods, the shape functions are defined (more or less) globally, independent of the interface. Therefore the shape functions overlap with the interface and adding this effect to the method negates the advantage of removing the mesh.

3.2.2 Lagrangian multipliers

As alternative to completely disregarding the mesh, we could also construct a mesh without considering the interfaces. The interfaces would then cut through the elements. This greatly simplifies the mesh generation as we do not have to accurately approximate the interfaces with the mesh.

Of course, this only shifts our problem, as we now have to impose our boundary conditions on an interface that intersects the elements. A possible solution is to introduce Lagrangian multipliers on the interface, see, for example, [11] (internal interfaces) and [18] (external interfaces).

Introducing Lagrangian multipliers exposes the difficulty of these methods for our problem. Our understanding of these methods is that they assume that the intersection of an element and the interface can be approximated by a flat surface or line. This is the complete opposite of our goal. Combining this with the already complex construction of the Lagrangian multiplier space for these flat intersections, we can only conclude that this approach does not seem promising for us.

3.2.3 XFEM

Instead of coupling using the Lagrangian multipliers, we can add some extra basis functions that can handle the interface that intersects the element. The idea of adding extra basis functions to handle all kinds of special behaviour is the defining property of Extended FEM (XFEM), see, for example, [34].

The benefit of adding extra basis functions is of course dependent on how well the basis functions match with the expected behaviour of the PDE. For example, if we theoretically expect a jump discontinuity over the interface, it can be beneficial to add a shape function that is 1 on one side of the interface and -1 on the other. Using such basis functions when we expect a discontinuity in the derivatives is of course far less useful.

The effect of adding extra basis functions depends on how well their shape matches the behaviour of the PDE. In our case, this depends on how well they suit the behaviour near the internal interfaces. Since we do not know what this behaviour is, in fact it is what we want to compute, it seems not a good idea to add these extra functions to our search space as a way to handle the interfaces. However, this could be an interesting addition in the future.

3.3 Polytopic elements in general

The previous methods all removed the requirement for generating a mesh that resolves the interface and their problems motivate why we want to mesh the interface in detail. Considering that the traditional element shapes result in a mesh with too many elements, we come to the conclusion that we need more general shapes. To allow both simple testing in 2D and the actual problem in 3D, we look for finite element methods that can work with polytopic element shapes.

Before looking at these methods, we recall the basic components of our reference method with some focus on the properties of the basis functions. Using this list of components we can get an overview what problems there are when using polytopic elements.

3.3.1 Properties of a traditional FEM

Our reference method has several components that allow the computation and approximation of the solution. These are

- a reference element $\hat{E}$,
- basis functions $\hat{\phi}_i$ defined on this element,
- a mesh $\mathcal{T}_h$ consisting of elements that are topologically equivalent to the reference element $\hat{E}$,
- an algorithm for mapping between an element $E \in \mathcal{T}_h$ and the reference element $\hat{E}$,
- an accurate quadrature rule on the reference element,

If we look at the k-th order basis functions in more detail, we see that they are not just any set of independent polynomials from $\mathcal{P}_k\left(\hat{E}\right)$. They are constructed such that they satisfy the following properties:

1. Kronecker delta: Each basis function corresponds to a single point $\hat{\mathbf{x}}_j$ on the reference element, such that $\hat{\phi}_i(\hat{\mathbf{x}}_j) = \delta_{ij}$.
2. Local sides: The only basis functions $\hat{\phi}_i$ which have support on a facet[1] e of the element are those basis functions that have their associated point on or adjacent to e.
3. Partition of unity: $\sum_i \hat{\phi}_i = 1$ anywhere on the reference element.

These properties can be found in many papers on interpolation based polygonal FEM (e.g.[53, 49]). Combining the first two properties with the standard distribution of the nodes $\hat{\mathbf{x}}_i$ allows for the construction of conforming basis functions, as basis functions on neighbouring elements with the same point coincide on the edge. In addition, this greatly simplifies imposing Dirichlet boundary conditions. The third condition ensures that we can represent a constant. In addition to these properties, we also have

4. Regularity: $\hat{\phi}_i$ is at least C^1 and in fact C^∞ on the element.
5. k-th order accurate: Given a polynomial function g of order not exceeding k, we can find coefficients g_i such that the interpolation $\bar{g} = \sum_i g_i \hat{\phi}_i(\mathbf{x})$ is exactly equal to g on the element.
6. Boundedness: each $\hat{\phi}_i$ is bounded from above and below on the reference element. Moreover, for $k = 1$ the lower bound is 0.

[1]A d dimensional polytope is bounded by several polytopes of dimension $d - 1$, which are called facets. For example, the 2D polytope, a polygon, is bounded by line segments, thus its facets are line segments. In three dimensions, a polytope is a polyhedra and its facets are polygons.

The regularity property ensures the existence of the derivatives needed in the bilinear form. The k-th order accuracy [46] ensures the convergence for the second order elliptic model problem that we use. The boundedness gives us a bounded solution, while the non-negativity of the basis functions for $k = 1$ allows an easy proof of the discrete maximum principle.

3.3.2 Problems with polytopic elements

With some small adaptions to the algorithm for simplexes, like the use of tensor product basis functions, we can adapt the method for quadrilaterals and cuboids. However, extending them to more general polytopes is far from trivial. We quickly summarize the problems.

Extending the reference element to polygons with more than 4 sides is no problem. For example [49] uses n equally spaced points on a unit circle to form a general reference element for a n-gon. We have not found any reference element for general elements in three or higher dimensions. We expect that the number of topological possibilities for an arbitrary polygon is far too high.

The basis functions are also a problem. The standard basis functions for simplexes and tensor product basis functions are both polynomials. When considering more general polytopes, we can no longer have polynomial basis functions that satisfy all previously mentioned properties [53]. Thus we are left with a choice: either sacrifice some of the properties or define non-polynomial basis functions.

We can not judge how complex the creation of a polytopic mesh is and it will depend on the problem. However, the mapping and quadrature rule on its elements are definitely a problem. Mapping convex polygons to a reference element is doable, as is illustrated in [49]. However, when we allow for concave polygons there is, to the best of our knowledge, no literature demonstrating a procedure for it, let alone when the element is polyhedral.

As the reference element seems to make everything harder, it is not surprising that none of the polytopic methods in literature use such an element. This removes the problem of defining a set of reference elements and mappings to them. Thereby also removing the option of defining the basis functions and quadrature on the reference element. Hence we need to define them in physical space.

Both the question of how to define basis functions and the quadrature do not have a simple solution. We will discuss the several alternatives from literature in the following sections.

3.4 Polytopic FEM

The first option, or actually family of options, is to try and keep as much of the traditional approach to polygonal or polyhedral elements to get Polytopic FEM (PFEM)[2]. Thus, we try and find basis functions that satisfy all of the properties from the Section 3.3.1 and then construct the bilinear and linear forms using some form of quadrature rule.

The hardest part is in extending the basis functions. The general trend in PFEM literature [32] is to restrict to extending the linear basis functions to polygons. Apart from that this is the

[2]Depending on the dimension of the problem the acronym can also stand for polygonal or polyhedral FEM, which are terms also used in literature.

simplest case, there is also another reason. The linear nodal basis functions coincide with the barycentric coordinates, which can be used to interpolate nodal data to the interior of triangles. Extending the ideas of barycentric coordinates to more sided polygons is therefore useful for more than finite element research and has a larger community working on it.

This is hardly a new approach as the first option for such an extensions was presented in 1971 by Wachspress [53]. However, interest of the wider research community is more recent, starting around 2003 (for example [26, 49]).

3.4.1 Overview of PFEM

The research in on these methods is hardly new, with the first examples presented in 1971 by Wachspress [53]. Interest in these methods remained low and has been renewed around 2003 (for example [26, 49]). Since then a plethora of generalized barycentric coordinates have been suggested. To get an impression of the possibilities we look at two review papers on this subject.

The review is by Sukumar and Malsch [48] from 2006. Its discusses several generalized barycentric coordinates which, due to the age of the review, can be characterized as the older coordinate choices. Interestingly it also tests these choices as linear basis functions for a simple Laplace problem. They show that the resulting error varies by several orders of magnitude, both between different methods on the same mesh and between different meshes with the same method.

The review by Floater [32] from 2015 is more extensive, discussing five different coordinate systems in detail with references to more. The most interesting point is the open question "For non-convex polygons and polyhedra, are there other coordinates that are smooth, positive and have simple closed form?", where other refers to the Harmonic coordinates. Coordinates found by solving the Laplace equation on the element with properties from Section 3.3.1 as constraints. Which are hideously expensive for our use, as they require solving the Laplace equation for each element multiple times.

Of all the presented coordinates in this review, only the Mean Value Coordinates can be extended to non-convex polyhedra. Moreover, the suggested construction of coordinates spanning the quadratic polynomials is expensive, as it requires constructing all pairs of basis functions $\phi_i\phi_j$. Limiting this is possible, but the only reference does it on convex polygons.

3.4.2 The options

From these reviews we get the impression that finding good coordinates is hard. Especially if we want to use them on non-convex polyhedra. In the literature we found three options, Mean Value Coordinates (MVC), Natural Element coordinates (NEC) and Maximum Entropy Coordinates (MEC).

The general working of each these coordinates is based on the same idea. To evaluate them at a point $\mathbf{x}$ in a polygon, we compute a weight $\lambda_i(\mathbf{x})$ corresponding to each vertex $\mathbf{x}_i$. Then by normalizing we find the value of each coordinate:

$$\phi_i(\mathbf{x}) = \frac{\lambda_i(\mathbf{x})}{\sum_{j=1}^{n_{E,v}} \lambda_j(\mathbf{x})}.$$

With the difference between them in how $\lambda_i(\mathbf{x})$ is defined.

For the MVC (2D [30], 3D [33]), they can be defined in a geometric way. However, a more practical method for 2D is derived in [31] and gives as weights:

$$\lambda_i(\mathbf{x}) = \sqrt{\|\mathbf{d}_{i-1}\|\,\|\mathbf{d}_{i+1}\| - \mathbf{d}_{i-1}\cdot\mathbf{d}_{i+1}} \prod_{j\neq i-1,i} \sqrt{\|\mathbf{d}_j\|\,\|\mathbf{d}_{j+1}\| + \mathbf{d}_j\cdot\mathbf{d}_{j+1}},$$

with $\mathbf{v}_i$ the coordinates of the $n_{E,v}$ vertices, $\mathbf{d}_i = \mathbf{v}_i - \mathbf{x}$ and circular indexing for $\mathbf{d}_i$. Even though this is probably more efficient than the original geometric interpretation, this formula still looks expensive, requiring a large number of square roots.

Both NEC and MEC do not fare better. The weights for NEC are defined by using a Voronoi diagram [41], which are quite expensive to construct. While the weights of MEC are determined by the maximizing Shannon's information entropy [47, 40], a non-linear function.

3.4.3 Summary

In trying to minimize the differences with the traditional finite element approach, PFEM forces itself into a corner. The proposed alternative conforming basis functions in the form of generalized barycentric coordinates are all expensive to compute. Especially when considering that we not only need their value at each quadrature point but also their gradients.

3.5 Virtual element method

In the previous section we demonstrated the problem of easy-to-compute basis functions that also satisfy the properties of the standard basis functions. The root cause of this problem is that the basis functions satisfying all the criteria on general polytopes are non-polynomial and as a result expensive to compute. Combine this with the cost of finding quadrature rules on polytopes, and we see that PFEM is computationally expensive.

This is in stark contrast to polynomials, which we can integrate over an arbitrary polytope without needing a quadrature rule for the polytope. We can do this by applying the divergence theorem to convert such an integral to a boundary integral, for example using a rule like

$$\int_E x_1^n \,\mathrm{d}\mathbf{x} = \frac{1}{n+1}\int_{\partial E} x_1^{n+1}\mathbf{n}_1 \,\mathrm{d}S, \tag{3.1}$$

with $n \neq -1$ and $\mathbf{n}$ the outward pointing normal. Doing this one or more times, we can rewrite the integral as an integral over the edges of the polytope, for which quadrature rules are available. Moreover, note that in doing this procedure, we never needed to compute the values of our polynomial inside the polytope, but only on its edges.

These ideas are at the basis of the Virtual Element Method (VEM). Its name derives from never having to compute the actual value of the basis functions and hence letting them remain 'virtual'. Originally, this method started as an offspring of the Mimetic Finite Difference methods into a finite element framework [12]. In the short span of time since this paper was published in 2012, it has taken flight, with several papers with both theoretical (e.g. [3, 14, 52]) and practical (e.g. [4, 17, 6]) focus.

Of course, this idea of letting the basis functions remain virtual, while still having a good finite element method is not a simple trick. The core ideas to achieve this are (quoting [12]):

- "The trial and test functions contain, on each element, all the polynomials of degree $\leq k$, plus some other functions that, in general, will not be polynomials."
- When computing, on each element, the local stiffness matrix (or rather the local stiffness bilinear form) we take particular care of the cases where one of the two entries is a polynomial of degree $\leq k$. The degrees of freedom are carefully chosen in order to allow us to compute the *exact* result using only the degrees of freedom of the other entry that in general will not be a polynomial.
- We can show that for the remaining part of the local stiffness bilinear form (when a non-polynomial encounters a non-polynomial) we only need to produce a result with *right order of magnitude and stability properties.*"

Here, $k \geq 1$ is the order of accuracy that we desire. The details on how to exactly implement these ideas are considerable, with the introduction using a simple Poisson problem problem in [12] and a how-do-we-do-this-numerically-guide in [13]. To introduce the method further, we will only give a basic idea, highlighting a few important points from these papers. For a better understanding, we highly recommend the aforementioned two papers.

3.5.1 Basic overview

To show how these core ideas are translated into the steps to solve a simple Poisson problem we will show the basic outline of VEM. Specifically, we look at the function space, degrees of freedom and how to construct the bilinear form. Note that we do this for a single element, as the procedure to get the global conforming basis functions is the same as for traditional basis functions. For more details, we again recommend [12, 13].

We start by defining the function space, the part that caused so much trouble for PFEM. For the definition of the element space we need two helper spaces, the usual $\mathcal{P}_k(E)$, the space of polynomials of degree at most k on a domain E, and the space of continuous polynomials on the boundary

$$\mathcal{B}_k(\partial E) = \left\{ v \in C^0(\partial E) \ : \ v|_e \in \mathcal{P}_k(e) \ \forall e \text{ edge of } \partial E \right\}.$$

Using this we define the finite element function space

$$V_k(E) = \left\{ v \in H^1(E) \ : \ v|_{\partial E} \in \mathcal{B}_k(\partial E), \Delta v|_E \in \mathcal{P}_{k-2}(E) \right\}. \tag{3.2}$$

Note that we can easily verify $\mathcal{P}_k(E) \subset V_k(E)$, as required by the first core principle.

For a function $v \in V_k(E)$ we define the following degrees of freedom:

- the values of v at the vertices of E,
- for $k > 1$, the values of v at $k-1$ predefined points on each edge e,
- for $k > 1$, the moments $\frac{1}{|E|}\int_E m(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x}$ for $m \in \mathcal{M}_{k-2}(E)$.

Note that these are all functionals $\chi_i : V_k(E) \to \mathbb{R}$, as prescribed by the theoretical definition of a finite element of Ciarlet [21]. Using this traditional definition and the requirement that the values of χ_i uniquely define a function $v \in V_k(E)$, one can construct the basis functions by requiring $\chi_i(\phi_j) = \delta_{ij}$.

An important observation is that for each edge e, the degrees of freedom specify the value of a function at $k+1$ points. By definition of $v \in V_k(E)$, we have that $v|_e \in \mathcal{P}_k(e)$, so $v|_e$

is completely determined by only these degrees of freedom and is only nonzero if one of these degrees of freedom is nonzero. The result is that for edge or vertex degrees of freedom for an edge $e = \bar{E}_1 \cap \bar{E}_2$, the corresponding basis functions ϕ_{E_1}, ϕ_{E_2} are identical on e, thus $\phi_{E_1}|_e = \phi_{E_2}|_e$. This matching at the edges allows the construction of global conforming basis functions.

Not only do the choice of the degrees of freedom allow for conforming elements they also allow the exact computation of the bilinear form for a polynomial (of degree at most k) and basis function. Specifically, for $p \in \mathcal{P}_k(E)$ we get

$$\mathcal{A}_1^E(p, \phi_i) = \int_E \nabla p \cdot \nabla \phi_i \,\mathrm{d}\mathbf{x} = -\int_E \phi_i \Delta p \,\mathrm{d}\mathbf{x} + \int_{\partial E} \phi_i \nabla p \cdot \mathbf{n} \,\mathrm{d}S. \tag{3.3}$$

Where we used $\mathcal{A}_1^E$ to denote that this is the contribution to the bilinear $\mathcal{A}$ for element E with $\kappa = 1$.

To compute the second integral exactly, we use the property that we can reconstruct ϕ_i exactly on each of the edges in ∂E. For the first integral on the right hand side, we use that $\Delta p \in \mathcal{P}_{k-2}(E)$ and we can therefore write it in terms of the scaled monomials $m_i \in \mathcal{M}_{k-2}(E)$. We can therefore write this integral as a linear combinations of integrals of the form $\int_E m_j \phi_i \,\mathrm{d}\mathbf{x}$, which allows computation via the moment degrees of freedom.

While this exact computation is used by [12] to prove existence and uniqueness of the solution, we also need it for approximating the bilinear form. To do this, we need the projection operator $\Pi^\nabla : V_k(E) \to \mathcal{P}_k(E)$, which is defined by that, for any $v \in V_k(E)$, we have

$$\mathcal{A}_1^E(p, v - \Pi^\nabla v) = 0 \qquad \forall p \in \mathcal{P}_k(E). \tag{3.4}$$

After a good choice to fix the constant part of $\Pi^\nabla v$, we can use of (3.3) allows the exact compute for each of the degrees of freedom.

We then continue by splitting the bilinear form:

$$\mathcal{A}_1^E(\phi_i, \phi_j) = \mathcal{A}_1^E(\Pi^\nabla \phi_i, \Pi^\nabla \phi_j) + \mathcal{A}_1^E(\phi_i - \Pi^\nabla \phi_i, \phi_j - \Pi^\nabla \phi_j),$$

where we used the definition of the projection operator to conclude that $\mathcal{A}_1^E\left(\Pi^\nabla \phi_i, \phi_j - \Pi^\nabla \phi_j\right) = 0$. Now applying the third core idea, we compute the first part exactly while the latter part is approximated and get

$$A^E = \mathcal{A}_1^E(\Pi^\nabla \phi_i, \Pi^\nabla \phi_j) + S_{ij}. \tag{3.5}$$

Where S_{ij} is an approximation of $\mathcal{A}_1^E(\phi_i - \Pi^\nabla \phi_i, \phi_j - \Pi^\nabla \phi_j)$, which should have the correct scaling behaviour to ensure the stability of the method. The original paper [12] provides a computationally very cheap version, while there are computationally more expensive versions that have better stability properties (e.g. [4]). As there is no A^e (2.14) needed for conforming basis functions, we have that (3.5) is sufficient for computing the stiffness matrix.

3.5.2 Broader view

This introduction looked at VEM for a Poisson problem. With a complicated method like VEM, the question arises, if we can extend this approach to different problems. To show there is more potential we look at some applications and extensions.

The method was, initially, applied to a plate bending problem [17], which is a fourth order problem. This requires some changes to the element space to get C^1 elements.

Time integration for a parabolic problem was tackled in [52]. Specifically, they solved the heat equation in 2D using backward Euler. Their results include good numerical tests and analytic bounds on the error.

The ideas of VEM lend well to using elements with special properties. Constructing an elements to be in C^l is possible using [25]. Additionally, $H(\text{div})$ and $H(\text{rot})$ conforming spaces were presented in [15] and extended by [24] to polyhedra.

The basic ideas of combining the Discontinuous Galerkin approach and VEM are presented [16], adjusting the SIPG stabilization terms to be computable with the VEM basis functions. Additionally proving that the method converges and providing standard error estimates.

3.5.3 Summary

We introduced the basic idea of VEM, a conforming method on polygonal and polyhedral elements that does not compute the basis functions inside the element. We showed the main steps how we can nevertheless approximate the bilinear form with sufficient accuracy to allow for the standard polynomial accuracy. The necessary steps do come at the price, they require more analysis and make the method more complex.

This complexity does not preclude extension beyond our simple example problem, as literature shows that VEM is both versatile and adaptable. Lastly, the need for analysis to show that the necessary approximations are sound, also ensures that there are solid analytical results on the method.

3.6 ADG and CFE

The third set of methods we look into consists of the Composite Finite Element method (CFE) and Agglomeration Discontinuous Galerkin Finite Element method (ADG). While different in origin, the more recent Discontinuous Galerkin formulation of both methods is practically identical.

Both methods try to solve the same problem as we have in our application: for a domain with fine details, the mesh size required for approximating them can become much smaller than the mesh size needed for the required accuracy of the solution. In other words, we need a mesh that is too fine when using standard elements to resolve the geometric details. Both CFE and ADG solve this by creating a coarser computational mesh that does not consist of simplexes. How they construct this coarser mesh is where the methods differ in the current interpretation.

3.6.1 ADG history

The history of ADG is not entirely clear. To our best interpretation, it seems that it is the result of two PhD projects.

The first project was by Tesini [51], who looked into creating h-Multigrid for DG algorithms on unstructured meshes. To create the necessary nested meshes, they agglomerate the elements of the finer meshes into larger elements. The resulting elements have rather arbitrary shapes and therefore require the use of specially constructed polynomial basis functions.

The second project was by Colombo [23], who looked at using the same basis functions as Tesini to create a new variant of DG with h-refinement using an agglomerated mesh. Thus now starting with a mesh of large agglomerated elements and then refining the mesh by adjusting the agglomeration algorithm to locally create smaller elements.

More recent works like [10, 19, 22] continue based on the latter idea. Starting with a very fine mesh, an agglomerated mesh is build using for example Metis [1] or MGridGen [2, 42], where the size of agglomeration is tuned based on error estimates.

3.6.2 CFE history

The alternative to ADG is the Composite Finite Element method (CFE). The major difference is the way the coarse mesh is generated. Where ADG just groups the elements together, CFE uses a hierarchy of meshes, the finest fitted to the small details and a more coarse one for the actual computations.

The reason for this complicated construction is that, with some effort, it allows for conforming basis functions [39, 38]. However, more recent developments seem to discard this rather complicated procedure and use discontinuous Galerkin methods instead (see, for example, [7, 5, 35]). These newer papers still mention the more complicated procedure to create basis functions by using prolongation and restriction operators on standard polynomial basis functions on the most refined mesh. The result of these complex procedures for the mesh and basis functions, can in the modern interpretation also be achieved much simpler by ADG.

For completeness we will sketch the main idea behind the mesh generation. This starts by creating a coarse mesh $\mathcal{R}$ that overlaps with the computational domain. This mesh is refined several times (depending on the size of the details) and is fitted to the domain by adjusting the nodes near the boundary and by removing elements that lie outside the domain. The result is a mesh that will have far too many small details. To construct the coarser mesh, we group all elements together that share the same ancestor in $\mathcal{R}$. Depending on the actual domain, the resulting mesh still resembles $\mathcal{R}$, as the only elements that are deformed are the ones that overlap with the details.

3.6.3 Current interpretation

While ADG and CFE differ in their mesh generation, the result is practically identical. They generate a coarse mesh $\mathcal{T}_h$ for computation, where each element consists of several triangles from a fine mesh $\mathfrak{T}_h$. In the current interpretation, they continue by using the same ideas for the actual method.

Since the mesh consists of rather arbitrary elements, we again need different basis functions. Here ADG and CFE make a different choice from PFEM and VEM, and keep the advantageous of polynomial basis functions from standard FEM. The cost in this case is that the resulting basis functions are not conforming, which is not a problem for methods like SIPG or the second method of Bassi and Rebay [10].

The creation of these basis functions is relatively simple. To create a basis for the polynomials of order at most k, we could use the scaled monomials. To improve their numerical behaviour we orthogonalize them using the modified Gramm-Schmidt algorithm. However, as shown by [51], this still results in rather large errors for higher order stretched elements. These effects are reduced by using monomials in a coordinate system adjusted to the shape of the element, using translation rotation and scaling to reduce the numerical errors.

For this combination of polynomial basis functions in combination with SIPG there is some analysis available. Moreover, by a specific choice of the SIPG stabilization term it can handle facets that are arbitrarily small, while maintaining the order of convergence.

3.6.4 Summary

We have introduced both CFE and ADG, which use polynomial basis functions on polytopic elements. CFE has some historical complexity that allows for constructing conforming basis functions. The more modern interpretation uses effectively the same basis as ADG, but requires a more complicated meshing procedure. Therefore we only consider ADG, as the method and implementation are simpler, while giving an almost identical method in a DG setting.

The characteristic property of both of these methods is the use of polynomial basis functions where the degrees of freedom are not coupled to the geometry of the element. As a consequence, the number of basis functions is coupled to the degree of the polynomials and not to the element shape. This lack of coupling has several advantages and disadvantages.

Starting with the advantages, the basis functions are polynomials. The basis construction is an easy normalization procedure, and we can integrate them exactly. Additionally, we choose the number of these basis functions independent of the shape and can therefore use polytopes with a significant number of facets without requiring a huge number of degrees of freedom.

The disadvantages is that since the degrees of freedom are no longer coupled to the vertices (or edges), we can not connect them between elements. The resulting nonconforming basis functions require the use of a DG method. Moreover, when coupling the degrees of freedom to vertices, the solution can be more accurate near rough regions with many vertices. Whether this lack of focus in these regions is a problem depends on the application.

3.7 Quadrature

So far, we have considered and reviewed the methods with a focus on the properties of their basis functions. However, for both the ADG/CFE and PFEM methods, we also need quadrature rules on their polytopic elements. This is a real problem, as we need such rules for non-convex polytopes of varying topology with possibly (for PFEM) non-polynomial integrands. Literature provides us with three different approaches.

The standard approach for PFEM [41, 48, 50] and ADG/CFE [9, 10, 20, 19] is the same. A polytope can be split into in simplexes, for which we have quadrature rules. Therefore, we can compute the integral over the polytope by applying quadrature on each simplex and then summing the result. This approach is especially suitable to meshes constructed in ADG/CFE as its elements are already composed of simplexes.

Since this requires evaluating the quadrature rule on each simplex, the method can become very expensive. Especially for agglomerated elements, which can consisting of many simplexes. Though expensive, it has advantageous too. It is simple to implement and analyze, and its accuracy can be adjusted by changing the quadrature rule. Moreover, it does not require any knowledge of integrand.

Alternatively, one can look into generating quadrature rules for the arbitrarily shaped polytopes. For example [43, 54] start with quadrature with many points and iteratively reduce the size while maintaining polynomial accuracy. In each iteration a point is removed, followed by adjusting the positions and weights of the remaining quadrature points to restore the accuracy. On the other hand, [44] constructs a quadrature by transforming a reference quadrature rule to the polygon. The used Schwarz-Christoffel mapping is constrained to 2D and computing it requires solving a set of non-linear equations. The common disadvantage is that both the optimization and construction of the mapping is a computationally expensive operation. Whether this extra cost is worth the reduction in quadrature points is of course dependent on the problem.

For polynomial integrands there is a third option, that is used by VEM. Using rules like (3.1) we can convert the element integral to an integral over the edges of an element. This increases the order of the polynomial integrand, but allows the use of standard quadrature on lines. Comparing the cost with the first alternative is dependent on the actual polytope. For this conversion approach we can use line quadrature, which are smaller in size than element quadratures. But a optimal subdivision in simplexes of a polytope usually contains fewer simplexes than edges.

3.8 Comparison

The previous discussion focussed mainly on the individual methods. As final part of this overview, we compare at all the available methods. For this, we use the criteria from Section 3.1.

We start by looking at both PFEM and VEM, as they associate degrees of freedom with each vertex and ensure that we have conforming basis functions, thereby extending the more traditional continuous Galerkin methods to polytopic elements. To achieve a conforming basis on polytopic elements they are both forced to use non-polynomial basis functions. However, the methods take completely different approaches for the problems related to differentiation and integration of them on the element. PFEM takes the traditional approach and uses numerical quadrature, while VEM takes the novel approach of carefully defining the basis functions such that we do not need to compute them and can approximate the non-polynomial part.

While this introduces some error due to the approximation, the analysis shows that the impact is acceptable. On all criteria we see that VEM is preferable to PFEM. Its main advantage is that we expect VEM to be computationally faster, as we do not have to compute the complicated basis functions. Moreover, there is a DG implementation, it supports higher order basis functions, and there is more literature on analysis of the method. Lastly the availability of specialized function spaces ($H(\text{div})$, $H(\text{curl})$, etc.) is a major benefit for our application.

While definition and use of the non-polynomial basis functions is done by VEM, ADG takes a fundamentally different route. It defines non-conforming but polynomial basis functions, which are much easier to differentiate and integrate. Moreover, the non-conforming basis functions require discontinuous Galerkin methods like SIPG.

This fundamental difference makes choose between VEM and ADG a hard problem. We expect both methods to be able to reduce the required degrees of freedom for the rough interface in the geometry of our application. ADG will, due to its lack of coupling between vertices and degrees of freedom, be able to further reduce the number of degrees of freedom when compared to VEM. But as noted in Section 3.6.4, it does not have the local variability near the interface.

Looking at our additional criteria, we also see no clear winner. Both methods can readily be extended to higher orders and can be coupled to traditional CG or DG methods on simplexes. We can safely say that VEM is not the easiest method to implement. However, given the extensiveness of the literature and the background from mimetic finite differences, this does seem a relatively minor problem. Lastly, for both ADG and VEM, there is literature on the analysis of the methods, containing both convergence and error estimates (e.g. [12]).

In conclusion, both ADG and VEM have their advantages and disadvantages, without a clear winner for our application. We expect both ADG and VEM to have lower computational cost and better prospect for future use, when compared to PFEM. We therefore select both ADG and the DG formulation of VEM (DGVEM) for further numerical tests to determine their potential for our intended application.

Chapter 4

ADG

As it is the simplest method of the two, we start by discussing how to implement ADG. When switching from physical space SIPG to ADG on triangles, we only have to change the construction of the basis functions. More work is, however, needed to switch the integration from using quadrature on triangles to an approach that works on polygons.

We could simply switch to triangulating the element and using quadrature on each of them. Such an approach will work for ADG, but will not work with the virtual basis functions of DGVEM. As these approaches can be beneficial for ADG we will introduce them here, with the added benefit, that the polynomial basis functions of ADG are easier to work with.

The chapter will therefore start with the theoretical discussion of the construction of the basis functions. Subsequently, we will discuss the numerical integration on polyhedra for the basis function construction, the element matrix, the element vector and error norm.

4.1 Basis construction

As already discussed in the previous chapter, the basis functions are the main difference between ADG and SIPG. For ADG these are constructed by orthogonalizing, for each element, a suitable starting set of the polynomials.

This orthogonalization is done using the modified Gramm Schmidt algorithm, possibly adding reorthogonalization to reduce the round off errors. As seen from Algorithm 1, this uses a set of initial monomials $\{\hat{m}_i\}_{i=1}^{n_{\mathcal{P},k}}$ on E, to build the actual basis functions ϕ_i.

The quality of the resulting basis depends on both the element shape and the starting set [10, 51]. The best results are for monomials of the form

$$\hat{m}_{\boldsymbol{\alpha}} = c_{\boldsymbol{\alpha}} \left[R(\mathbf{x} - \mathbf{x}_E)\right]^{\boldsymbol{\alpha}},$$

where R is a rotation matrix only depending on the element and $c_{\boldsymbol{\alpha}}$ is such that the monomial has an $L^2(E)$ norm of 1. The rotation matrix is to compensate for possible geometric anisotropy, for example by aligning axis of the coordinate system defined by $\mathbf{x} - \mathbf{x}_E$ with the principal moments of inertia of the element E.

Algorithm 1: The Modified Gramm-Schmidt algorithm used to construct the basis functions. Optionally one can add reorthogonalization by executing the orthogonalization loop (lines 3–5) twice. Note that, in addition to adding polynomials and scaling them, we also need to compute the L^2-inner product on the element E.

```
1 for $i \leftarrow 1$ to $n_{\mathcal{P},k}$ do
2     $\phi \leftarrow \hat{m}_i$
3     for $j \leftarrow 1$ to $i-1$ do
4         $\phi \leftarrow \phi - (\phi_j, \phi)_E \phi_j$
5     end
6     $\phi_i \leftarrow \frac{\phi}{\sqrt{(\phi,\phi)_E}}$
7 end
```

The effect of the choice of the starting set is especially important when using elements with large aspect ratios and higher order basis functions. As we do not intend to use such elements[1], we make life easier by reusing the scaled monomials that VEM needs. Note that this corresponds to the choice $R = I$ and $c_{\boldsymbol{\alpha}} = h_E^{-|\boldsymbol{\alpha}|}$. This does not give an $L^2(E)$-norm of 1, but ensures that the monomials are roughly of the same order of magnitude.

Having defined the starting set, we can almost perform the algorithm. We still need a way to compute the L^2-inner product on each element E. For triangular, elements this is no problem, as we can simply use the same quadrature rules as SIPG uses for triangles and write

$$(\phi, \phi_i)_E = \int_E \phi\phi_i \,\mathrm{d}\mathbf{x} \approx \sum_{(\mathbf{x}_q, w_q) \in \mathcal{Q}_{2k}(E)} \phi\phi_i \,|J|\, w_q\big|_{\mathbf{x}=\mathbf{x}_q}. \tag{4.1}$$

4.2 Quadrature-less basis construction

To extend the procedure to polygons, we only need to adjust the computation of the L^2-inner product in the algorithm. We could do this by triangulating the element and using quadrature on each triangle. However, as discussed in Section 3.7, this can be expensive. Instead, we will look at the alternative approach also used in VEM.

To evaluate the integral over an element for integrand g, we use the divergence theorem and write it as a boundary integral:

$$\int_E g \,\mathrm{d}\mathbf{x} = \int_{\partial E} G \cdot \mathbf{n} \,\mathrm{d}S, \tag{4.2}$$

where G is a vector valued function, such that $\nabla \cdot G = g$. For polygonal elements, we have that ∂E are its edges, for which we can use Gauss-Legendre or similar quadrature rules.

In this general setting, this seems to be a rather complicated procedure where we just traded the problem of an expensive quadrature rule for the problem of finding the function G. However, for the basis construction, we only use the integral to compute the inner product of two polynomial

[1]This decision was made relatively early in the implementation. The later developed mesh construction using tiles does sometimes result in large aspect ratio elements, fortunately these are at the boundary and usually aligned with the coordinate system.

functions. Thus, the function g is a polynomial of degree at most $2k$, for which we can easily find such a G. In fact, we can require that G is a polynomial of degree at most $2k+1$, allowing us to choose a quadrature rule that integrates $G \cdot \mathbf{n}$ exactly on each of the edges of ∂E.

To derive the actual procedure, let p, q be two arbitrary polynomials in $\mathcal{P}_k(E)$, for example ϕ and ϕ_i from Algorithm 1. We expand these in terms of the scaled monomials, and we denote the vectors with coefficients of p and q by $\mathbf{p}$ and $\mathbf{q}$, respectively. Using these vectors we write

$$\begin{aligned}(p,q)_E &= \int_E pq \,\mathrm{d}\mathbf{x},\\ &= \sum_{i=1}^{n_{\mathcal{P},k}} \sum_{j=1}^{n_{\mathcal{P},k}} \mathbf{p}_i \mathbf{q}_j \int_E m_i m_j \,\mathrm{d}\mathbf{x},\\ &= \mathbf{p}^T \mathcal{I}_E \mathbf{q},\end{aligned}$$

with

$$\mathcal{I}_{E,ij} = \int_E m_i m_j \,\mathrm{d}\mathbf{x} = \int_E \left(\frac{\mathbf{x}-\mathbf{x}_E}{h_E}\right)^{\mathbf{s}_i+\mathbf{s}_j} \mathrm{d}\mathbf{x}. \tag{4.3}$$

To evaluate these integrals, we define $\mathbf{s} = \mathbf{s}_i + \mathbf{s}_j$ and change coordinates to get unscaled monomials:

$$\int_E \left(\frac{\mathbf{x}-\mathbf{x}_E}{h_E}\right)^{\mathbf{s}} \mathrm{d}\mathbf{x} = \int_{\hat{E}} \mathbf{y}^{\mathbf{s}} \,|J| \,\mathrm{d}\mathbf{y} = {h_E}^2 \int_{\hat{E}} \mathbf{y}^{\mathbf{s}} \,\mathrm{d}\mathbf{y}, \tag{4.4}$$

where $\hat{E}$ is the transformed element E and $|J|$ is the Jacobian of the transformation $\mathbf{y} = h_E^{-1}(\mathbf{x}-\mathbf{x}_E)$.

Now it is time to apply the main idea of the procedure: convert the integral to a boundary integral using the rule

$$\int_{\hat{E}} \mathbf{y}^{\mathbf{s}} \,\mathrm{d}\mathbf{y} = \frac{1}{d+|\mathbf{s}|} \int_{\partial\hat{E}} \mathbf{y}^{\mathbf{s}} \mathbf{y} \cdot \mathbf{n} \,\mathrm{d}S. \tag{4.5}$$

With $d = 2$ the dimension of $\hat{E}$ and $\mathbf{n}$ the outward pointing normal vector. Applying this to (4.4) and using quadrature on each of the edges we get

$$\begin{aligned}\mathcal{I}_{E,ij} &= h_E^2 \int_{\hat{E}} \mathbf{y}^{\mathbf{s}} \,\mathrm{d}\mathbf{y} &(4.6)\\ &= \frac{h_E^2}{d+|\mathbf{s}|} \int_{\partial\hat{E}} \mathbf{y}^{\mathbf{s}} \mathbf{y} \cdot \mathbf{n} \,\mathrm{d}S, &(4.7)\\ &= \frac{h_E^2}{d+|\mathbf{s}|} \sum_{e \in \partial\hat{E}} \int_e \mathbf{y}^{\mathbf{s}} \mathbf{y} \cdot \mathbf{n} \,\mathrm{d}S, &(4.8)\\ &= \frac{h_E^2}{d+|\mathbf{s}|} \sum_{e \in \partial\hat{E}} \sum_{(\mathbf{y}_q, w_q) \in \mathcal{Q}_{2k+1}(e)} \mathbf{y}^{\mathbf{s}} \mathbf{y} \cdot \mathbf{n} w_q \big|_{\mathbf{y}=\mathbf{y}_q}. &(4.9)\end{aligned}$$

Remark. *While the integration rule* (4.5) *has the nice property that it has the lowest possible polynomial order, this is not the only rule with this property. We do not know if using one of the alternatives would significantly impact the numerical accuarcy or performance.*

4.2.1 Basis construction revised

Using the matrix $\mathcal{I}_E$, we can now reformulate the basis construction, Algorithm 1, for actual implementation. We do this by writing each of the basis functions in terms of the scaled monomial basis

$$\phi_j = \sum_{i=1}^{n_{\mathcal{P},k}} b_{ij} m_i.$$

We group all the coefficients b_{ij} into the matrix B. We compute its values using Algorithm 2, which is a simple translation of Algorithm 1.

Algorithm 2: The actual Modified Gramm-Schmidt algorithm used for the construction of the matrix B for a single element E. The vector e_i is the i-th unit vector (as column) and $B_{\star j}$ is the j-th column of the matrix B, containing all the coefficients of basis function ϕ_j.

```
1 Precompute $\mathcal{I}_E$
2 for $i \leftarrow 1$ to $n_{\mathcal{P},k}$ do
3     $\phi \leftarrow e_i$
4     for $j \leftarrow 1$ to $i-1$ do
5         $\phi \leftarrow \phi - \left(B_{\star j}^T \mathcal{I}_E \phi\right) B_{\star j}$
6     end
7     $B_{\star i} \leftarrow \phi / \sqrt{\phi^T \mathcal{I}_E \phi}$
8 end
```

4.3 Quadrature-less element matrix

With the matrix $\mathcal{I}_E$, but we can not only compute the basis functions efficiently, we can also use it to compute the local element matrix (2.13). Writing κ_E for the value of κ on E, we expand the local element matrix as

$$\begin{aligned} A_{ij}^E &= \kappa_E \int_E \nabla\phi_i \cdot \nabla\phi_j \,\mathrm{d}\mathbf{x}, \\ &= \kappa_E \int_E \frac{\partial\phi_i}{\partial x_1}\frac{\partial\phi_j}{\partial x_1} + \frac{\partial\phi_i}{\partial x_2}\frac{\partial\phi_j}{\partial x_2} \,\mathrm{d}\mathbf{x}, \\ &= \kappa_E \left[\left(\frac{\partial\phi_i}{\partial x_1}, \frac{\partial\phi_j}{\partial x_1} \right)_E + \left(\frac{\partial\phi_i}{\partial x_2}, \frac{\partial\phi_j}{\partial x_2} \right)_E \right]. \end{aligned}$$

From which we can see that we have to compute two inner products between the polynomial derivatives of the basis functions. To use $\mathcal{I}_E$ for this, we need to compute the scaled monomial coefficients of these derivatives.

As both $\frac{\partial}{\partial x_1}$ and $\frac{\partial}{\partial x_2}$ can be interpreted as linear operators of the form $\mathcal{P}_k(E) \to \mathcal{P}_k(E)$, we can represent them as two matrices D_{x_1} and D_{x_2} for use with the scaled monomial basis. Using these two matrices and the matrix for the basis function coefficients (B), the local element matrix can be written as

$$A^E = \kappa B^T \left[D_{x_1}^T \mathcal{I}_E D_{x_1} + D_{x_2}^T \mathcal{I}_E D_{x_2} \right] B.$$

To construct the differentiation matrices D_{x_1} and D_{x_2}, we will first consider the similar matrices for the unscaled monomials of the form $\mathbf{y}^{\mathbf{s}}$. Note that we can transform the scaled monomials into the unscaled ones using the transformation

$$\mathbf{y} = \frac{\mathbf{x} - \mathbf{x}_E}{h_E}. \tag{4.10}$$

Constructing the differentiation matrix D_{y_1}, corresponding to $\frac{\partial}{\partial y_1}$, is trivial using standard calculus:

$$\frac{\partial}{\partial y_1}\mathbf{y}^{(s_1,s_2)} = \frac{\partial}{\partial y_1} y_1^{s_1} y_2^{s_2} = s_1 y_1^{s_1-1} y_2^{s_2} = s_1 \mathbf{y}^{(s_1-1,s_2)}.$$

As each row in D_{y_1} corresponds to the mapping of a monomial $\mathbf{y}^{(s_1,s_2)}$ we have to put the value s_1 in the column for $\mathbf{y}^{(s_1-1,s_2)}$ (assuming $s_1 > 0$). This can be done using the pairing function from (2.3).

After an analogous construction of the matrix D_{y_2}, corresponding to $\frac{\partial}{\partial y_2}$, we still have to transform back to the matrices D_{x_1} and D_{x_2}. Applying standard derivation for the change of variables of (4.10) gives

$$\begin{pmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial y_1} \\ \frac{\partial}{\partial y_2} \end{pmatrix} = \frac{1}{h_E} \begin{pmatrix} \frac{\partial}{\partial y_1} \\ \frac{\partial}{\partial y_2} \end{pmatrix}$$

Hence, we have $D_{x_1} = \frac{1}{h_E} D_{y_1}$ and $D_{x_2} = \frac{1}{h_E} D_{y_2}$. Note that D_{x_1} and D_{x_2} are dependent on the element, while D_{y_1} and D_{y_2} are element independent, and thus only need to be computed once.

Remark. *Note that in 2D the contribution of the edges, A^e, can be computed using standard quadrature rules for lines. Therefore no special handling is needed. For problems in 3D, where A^e corresponds to face integrals, it becomes more complicated as the faces can be polygons.*

4.4 Efficient element vector calculation

Now that we have removed the need for an element quadrature in the basis construction and element matrix, we might also consider the possibility of removing it from the element vector F^E (2.15). Unfortunately, this is far more complicated, as f is an arbitrary function that, in general, is not polynomial on each element. Therefore we look at three alternatives for using the traditional triangulation and quadrature approach.

4.4.1 Conversion to boundary integral

The first alternative is to use the same approach as with polynomials: applying the divergence theorem to convert the integral over the element to a boundary integral, and then computing the boundary integral using standard line quadrature.

Applying this approach directly to the integral needed for the element vector,

$$F_i^E = \int_E \phi_i f \, \mathrm{d}\mathbf{x}, \tag{4.11}$$

is not so practical. We would have to find a vector function G such that $\nabla \cdot G = \phi_i f$, which is quite hard, as we only know that ϕ_i is polynomial. Instead, we use the fact that we constructed ϕ_i in terms of the scaled monomials and rewrite the integral as

$$F_i^E = \sum_j B_{ji} \int_E m_j f \,\mathrm{d}\mathbf{x}. \tag{4.12}$$

Hence, we only need to compute the moments $\tilde{f}_j$ of f with respect to the scaled monomials m_j. Using the divergence theorem gives

$$\tilde{f}_j = \int_E m_j f \,\mathrm{d}\mathbf{x} = \int_{\partial E} G_j \cdot \mathbf{n} \,\mathrm{d}S, \tag{4.13}$$

for any function G_j such that $\nabla \cdot G_j = m_j f$. Given such a function, we can use quadrature on each of the edges in ∂E to compute the moments $\tilde{f}_j$. Interpreting these moments as a vector $\tilde{\mathbf{f}}$ we can rewrite (4.12) to

$$F^E = B^T \tilde{\mathbf{f}}. \tag{4.14}$$

Thus, assuming that we can find the functions G_j, we can apply this approach. To further investigate the complexity in this approach we apply it to an example, and choose

$$f(\mathbf{x}) = \sin(2\pi x_1)\sin(2\pi x_2). \tag{4.15}$$

Plugging this into (4.13) gives

$$\int_E m_j f \,\mathrm{d}\mathbf{x} = \int_E \left(\frac{x_1 - x_{E,1}}{h}\right)^{s_1} \left(\frac{x_2 - x_{E,2}}{h}\right)^{s_2} \sin(2\pi x_1)\sin(2\pi x_2)\,\mathrm{d}x_1\,\mathrm{d}x_2, \tag{4.16}$$

where s_1 and s_2 are the powers for the j-th monomial. We then perform the change of basis $\mathbf{y} = \frac{\mathbf{x} - \mathbf{x}_E}{h_E}$ to get

$$\int_E m_j f \,\mathrm{d}\mathbf{x} = h_E^2 \int_{\hat{E}} y_1^{s_1} y_2^{s_2} \sin\left(2\pi(y_1 h + y_{E,1})\right) \sin\left(2\pi(y_2 h + y_{E,2})\right) \mathrm{d}y_1\,\mathrm{d}y_2. \tag{4.17}$$

The factor h_E^2 comes from the Jacobian introduced by the transformation from the element in physical coordinates E to one in the local coordinates $\hat{E}$. For easier notation, we split the integrand into two parts:

$$\begin{aligned} Y_{1,n}(y_1) &= y_1^n \sin\left(2\pi(y_1 h_E + y_{E,1})\right), \\ Y_{2,n}(y_2) &= y_2^n \sin\left(2\pi(y_2 h_E + y_{E,2})\right). \end{aligned}$$

Using these functions, we can rewrite the integral and apply the divergence theorem to obtain

$$\int_E m_j f \,\mathrm{d}\mathbf{x} = h_E^2 \int_{\hat{E}} Y_{1,s_1}(y_1) Y_{2,s_2}(y_2)\,\mathrm{d}y_1\,\mathrm{d}y_2 = \int_{\partial\hat{E}} G_j \cdot \mathbf{n}\,\mathrm{d}y_1\,\mathrm{d}y_2.$$

where G_j is the same vector function as in (4.13), which, in this example, should satisfy $\nabla \cdot G_j = Y_{1,s_1} Y_{2,s_1}$. The simplest form for G_j is a vector with only one non-zero component, for which we arbitrarily choose the first one. Thus, $G_j = \langle Y_{2,s_2} H_{s_1}, 0\rangle^T$ with $\frac{\partial}{\partial y_1} H_n = Y_{1,n}$, with the recursive solution

$$\begin{aligned} H_n(y_1) &= -\frac{y_1^n}{2\pi h_E}\cos\left(2\pi(y_1 h_E + x_{E,1})\right) + \frac{n}{2\pi h_E}\hat{H}_{n-1}(y_1), \\ \hat{H}_n(y_1) &= \frac{y_1^n}{2\pi h_E}\sin\left(2\pi(y_1 h_E + x_{E,1})\right) - \frac{n}{2\pi h_E} H_{n-1}(y_1). \end{aligned}$$

By combining all these steps, we get the final expression for the moments:

$$\tilde{f}_j = \int_E m_j f \,\mathrm{d}\mathbf{x} \approx {h_E}^2 \sum_{e \in \partial \hat{E}} \sum_{(\mathbf{y}_q, w_q) \in \mathcal{Q}_{k'}(e)} H_{s_1}(y_1) Y_{2,s_2}(y_2) n_1 w_q |_{\langle y_1, y_2 \rangle^T = \hat{\mathbf{y}}_q}, \tag{4.18}$$

where n_1 is the first component of the outward pointing normal vector and $\langle s_1, s_2 \rangle^T = \boldsymbol{\alpha}_j$ are the exponents of the j-th monomial. Furthermore, $k' \geq 2k$ is the quadrature order used to approximate the integral.

Looking at the example, we see that the complexity of this approach is in finding G_j and the actual computation of the moments $\tilde{f}_j$ that follows. In this example the computation looks relatively complex and expensive. For the actual computation of the complete moment vector $\tilde{\mathbf{f}}$, several optimizations are possible by reusing the values of H_n, $\hat{H}_n$ and Y_{2,s_2} between different moments, reducing the computational cost slightly.

4.4.2 Polynomial projection

As second alternative for the element vector computation, we approximate f by projecting it to a polynomial of degree at most l. Specifically we will compute the L^2-orthogonal projection $\Pi_l f$ of f onto $\mathcal{P}_l(E)$, followed by using

$$F_i^E = \int_E \phi_i f \,\mathrm{d}\mathbf{x} \approx \int_E \phi_i \Pi_l f \,\mathrm{d}\mathbf{x}. \tag{4.19}$$

This we can compute using $\mathcal{I}_E$ as both integrands are polynomial.

The L^2-orthogonal projection operator $\Pi_l : L^2(E) \to \mathcal{P}_l(E)$ is defined as

$$(\Pi_l f, p)_E = (f, p)_E, \qquad \forall p \in \mathcal{P}_l(E). \tag{4.20}$$

For its computation, we start, as usual, by expanding the result $\Pi_l f$, a polynomial, in terms of the scaled monomial basis:

$$\Pi_l f = \sum_i^{n_{\mathcal{P},l}} \alpha_i m_i.$$

Furthermore, we set $p = m_j$ in (4.20) for $j = 1, 2, \ldots, n_{\mathcal{P},k}$ and write the resulting equations as linear system:

$$\begin{pmatrix} (m_1, m_1)_T & \cdots & (m_1, m_{n_{\mathcal{P},l}})_T \\ \vdots & & \vdots \\ (m_{n_{\mathcal{P},l}}, m_1)_T & \cdots & (m_{n_{\mathcal{P},l}}, m_{n_{\mathcal{P},l}})_T \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{n_{\mathcal{P},l}} \end{pmatrix} = \begin{pmatrix} (m_1, f)_T \\ \vdots \\ (m_{n_{\mathcal{P},l}}, f)_T \end{pmatrix}. \tag{4.21}$$

Note that on the left hand side we have the matrix $\mathcal{I}_E$ from (4.3), while the right hand side we have the moments of f. We could apply standard triangulation and quadrature to compute these. Alternatively, we can reuse the computations from the previous section and continue with (4.13). Denoting the resulting moments of f by the vector $\tilde{\mathbf{f}}$, we have

$$\Pi_l f = \left(\mathcal{I}_E^{l,l}\right)^{-1} \tilde{\mathbf{f}}, \tag{4.22}$$

where $\mathcal{I}_E^{n,m}$ is the $n_{\mathcal{P},n}$ by $n_{\mathcal{P},m}$ matrix of inner products between scaled monomials:

$$\mathcal{I}_{E,ij}^{n,m} = (m_i, m_j)_E.$$

Using this in (4.19) we get

$$F^E = B^T \mathcal{I}_E^{k,l} \left(\mathcal{I}_E^{l,l}\right)^{-1} \tilde{\mathbf{f}}. \tag{4.23}$$

For the special case that $k = l$, we have that (4.19) is exact (set $p = \phi_i$ in (4.20)). Moreover, (4.23) can then be simplified:

$$F^E = B^T \mathcal{I}_E^{k,k} \left(\mathcal{I}_E^{k,k}\right)^{-1} \tilde{\mathbf{f}} = B^T \tilde{\mathbf{f}}.$$

Comparing this last formula to the result from the previous section (4.14), we see that they are identical. Using the polynomial projection as idea behind this result has two advantageous.

Firstly, we require less knowledge about the basis functions. In this second approach we only require from the basis functions the ability to compute the inner product with our approximation of f, a polynomial of degree at most l. This is far less demanding than in the first alternative, where we need to directly compute the inner product between the basis functions and the non-polynomial f.

Secondly, we can change the approximation order by setting $l \neq k$. When setting $l < k$, we reduce the number of moments that we have to compute, thereby reducing the computational cost, but loosing accuracy. Higher order approximations $l > k$ are not so useful here, as the case $l = k$ is already exact, but they can be useful when used in combination with the error norm computation.

4.4.3 Taylor polynomial

In the alternative of the previous section, we made a complex and expensive computation to approximate f on each element by a polynomial. Requiring both the inversion of $\mathcal{I}_E$ and the computation of the moments (4.23). To reduce the cost while still approximating f by a polynomial, we look at using the Taylor polynomial of f for the element vector computation.

The Taylor polynomial of f around the centroid $\mathbf{x}_E$ of the element is defined as

$$f(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \leq k} \frac{1}{\boldsymbol{\alpha}!} D^{\boldsymbol{\alpha}} f(\mathbf{x}_E)(\mathbf{x} - \mathbf{x}_E)^{\boldsymbol{\alpha}} + O\Big((\mathbf{x} - \mathbf{x}_E)^{k+1}\Big),$$

where $\boldsymbol{\alpha}! = \alpha_1!\alpha_2!$ and $D^{\boldsymbol{\alpha}} f = \frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2}}$. With some rearranging, we can express it in terms of the scaled monomials:

$$f(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \leq k} \frac{h_E^{|\boldsymbol{\alpha}|}}{\boldsymbol{\alpha}!} D^{\boldsymbol{\alpha}} f(\mathbf{x}_E) \left(\frac{\mathbf{x} - \mathbf{x}_E}{h_E}\right)^{\boldsymbol{\alpha}} + O\Big((\mathbf{x} - \mathbf{x}_E)^{k+1}\Big).$$

From which it directly follows that the coefficient for the j-th scaled monomial is given by

$$r_j = \frac{h_E^{|\boldsymbol{\alpha}_j|}}{\boldsymbol{\alpha}_j!} D^{\boldsymbol{\alpha}_j} f(\mathbf{x}_E).$$

After computing these coefficients and putting them into a vector $\mathbf{r}$, we can take the same approach as with the moments in the polynomial projection and approximate the element vector with

$$F^E \approx B^T \mathcal{I}_E \mathbf{r}. \tag{4.24}$$

When applying it again to the right hand side for our numerical test case (4.15) we get

$$r_{s_1,s_2} = \frac{(2\pi h_E)^{(s_1+s_2)}}{(s_1+s_2)!} \begin{cases} (-1)^{\left(\frac{s_1}{2}+\frac{s_2}{2}\right)} \sin(2\pi x_1)\sin(2\pi x_2) & \text{for } s_1, s_2 \text{ even;} \\ (-1)^{\left(\frac{s_1-1}{2}+\frac{s_2}{2}\right)} \cos(2\pi x_1)\sin(2\pi x_2) & \text{for } s_1 \text{ odd}, s_2 \text{ even;} \\ (-1)^{\left(\frac{s_1}{2}+\frac{s_2-1}{2}\right)} \sin(2\pi x_1)\cos(2\pi x_2) & \text{for } s_1 \text{ even}, s_2 \text{ odd;} \\ (-1)^{\left(\frac{s_1-1}{2}+\frac{s_2-1}{2}\right)} \cos(2\pi x_1)\cos(2\pi x_2) & \text{for } s_1, s_2 \text{ odd;} \end{cases}$$

where the $\mathbf{r}_i$ follow from $r_{s_1,s_2}$by using the standard multi-index order (2.4). From an implementation point of view, this expression looks both less expensive and more easy to implement than that for the moments (4.18) needed for polynomial projection. However, the use of Taylor polynomials gives rise to the question whether f has sufficient derivatives and whether the approximation is accurate enough. While the answer for this question ultimately depends on the exact function f, we can consider the following.

The function f is usually defined by the model or by the manufactured solution. Such functions usually have a more direct way to compute them. When it is available in the form of an expression using standard functions (polynomials, sin, log, etc.), it is usually easier to find the derivatives than finding anti derivatives. Thus in such case we expect that it is easier to construct the Taylor polynomial than to find the functions G_i needed for the polynomial projection.

For the accuracy we see that the error is of order $\mathcal{O}\Big((\mathbf{x}-\mathbf{x}_E)^{k+1}\Big)$. As $\mathbf{x}-\mathbf{x}_E < h_E \leq h$ we have an error of order h^{k+1} [20], which should be sufficient for small h. Whether this also holds for larger elements is, of course, dependent on f.

4.4.4 Comparison

In this section we looked at computing the element vector. The traditional approach in ADG is to use the costly triangulation and quadrature approach. Therefore, we proposed three different alternatives.

The first two alternatives computed the moments of f, using the divergence theorem and quadrature on the edges of the element E. This required finding functions G_j such that $\nabla \cdot G_j = f m_j$. Such functions can be hard or impossible to find in closed form or might be expensive to evaluate.

The third alternative approximated f by a Taylor polynomial, which, depending on f, can be easier to find than the functions G_j. Moreover, we only evaluate this at a single point, instead of the multiple points for integrating over the boundary. While this approach is computationally inexpensive, the question is how accurate the Taylor polynomial is.

Making a comparison at this point is almost pure speculation, as the results will depend on the specific f. Having said that, we do expect that the Taylor polynomial (if possible) may be an interesting alternative to use. The polynomial projection is probably less interesting, as the moments of f can be computationally expensive to compute.

4.5 Error computation

We do not only need the element quadrature to compute the element vector, we also need it for computing the error in the solution, $\|u - u_h\|_\Omega$. Just like with the element vector, the problem here is that the unknown function u does not need to be polynomial on each element. We therefore also look at adapting the approaches for the element vector computation to the computation of the local error

$$\|u - u_h\|_E = \int_E (u - u_h)^2 \,\mathrm{d}\mathbf{x},$$

which, after summation over all elements $E \in \mathcal{T}_h$ and taking the square root, gives the value of $\|u - u_h\|_\Omega$.

4.5.1 Direct computation

Adapting the first alternative used for the element vector (Section 4.4.1), we can again try to compute the integral directly using the divergence theorem. However, finding a function G, such that $\nabla G = (u - u_h)^2$ is expected to be even harder than finding the G_j for f. We therefore suspect that it is needed to split the integral into three parts:

$$\int_E u^2 \,\mathrm{d}\mathbf{x} - \int_E 2uu_h \,\mathrm{d}\mathbf{x} + \int_E u_h^2 \,\mathrm{d}\mathbf{x}.$$

Looking at the three integrals, we see three different types, for which three different approaches could be useful. The last integral involves the polynomial basis functions and can therefore be computed using $\mathcal{I}_E$ or even faster using the orthogonality of the basis functions. The second integrand is the product of a polynomial and a non-polynomial function, for which we can take the exact same routes as used with the element vector. The first integral is dependent on the actual choice of u, without the explicit form there is not much we can say about it.

While this approach is possible, it is rather complex. For each of the three terms we take a different approach with different error characteristics. Additionally, we are subtracting (large) values from each other, which might result in larger numerical errors. Even if we only require the first few digits of $\|u - u_h\|_E$ to be correct, this could still require large precision in each of the integrals. Moreover, we do not know how much accuracy is needed beforehand.

4.5.2 Polynomial projection

The second alternative from the element vector is to project the non-polynomial function, in this case u, onto the polynomials using Π_k. Which we can compute (just as with f) either using triangulation and quadrature, or using the divergence theorem, convert it into a boundary integral. Using triangulation and quadrature does not seem to be useful in this case as we could then also use it on the actual error $u - u_h$.

For this polynomial projection approach, we write $u = \Pi_k u + (u - \Pi_k u)$ to obtain

$$\int_E (u - u_h)^2 \,\mathrm{d}\mathbf{x} = \int_E (\Pi_k u - u_h)^2 \,\mathrm{d}\mathbf{x} + \int_E (u - \Pi_k u)^2 \,\mathrm{d}\mathbf{x}, \tag{4.25}$$

where we used that $\int_E (u - \Pi_k u) u_h \, d\mathbf{x} = 0$, as it corresponds to choosing $p = u_h$ in (4.20). As we can not compute the second integral of (4.25) we have to discard it to get

$$\int_E (u - u_h)^2 \, d\mathbf{x} \geq \int_E (\Pi_k u - u_h)^2 \, d\mathbf{x}.$$

Computation is then straightforward. Both $\Pi_k u$ and u_h are polynomials that we represent using the coefficients in the scaled monomial basis. Let $\mathbf{u}_d$ be the coefficients of $\Pi_k u - u_h$. We then get

$$\int_E (\Pi_k u - u_h)^2 \, d\mathbf{x} = \mathbf{u}_d^T \mathcal{I}_E \mathbf{u}_d. \tag{4.26}$$

Unlike the previous method we do not expect trouble of large rounding errors. Moreover, as the last integration step using $\mathcal{I}_E$ is exact, we can at least expect non-negative results. In contrast with these advantages, the part $u - \Pi_k u$ that we disregard is of order h^{k+1}, similar to the error that we expect in the solution. With possibly even more errors from the computation of $\Pi_k u$ affecting the actual error.

Though, one has to realise that even with the traditional approach using quadrature (2.24), one only approximates the value of $\|u - u_h\|_E$. To reduce the error from this approximation, we can use higher order quadrature rules for the local error. Similarly, we can use, for this polynomial projection approach, a higher order projection $\Pi_l u$, $l > k$, for computing the error. Though this comes at the cost of having to compute a larger $\mathcal{I}_E$ matrix that works for polynomials up to order l.

Remark. *When we can integrate u^2 over Ω or E, it is possible to compute the disregarded term exactly using*

$$\int_E (u - \Pi_k u)^2 \, d\mathbf{x} = \int_E u^2 + (\Pi_k u)^2 - 2u\Pi_k u \, d\mathbf{x} = \int_E u^2 - (\Pi_k u)^2 \, d\mathbf{x}.$$

We used that $(u, q)_E = (\Pi_k u, q)_E$ for all $q \in \mathcal{P}_k(E)$ and thus also for $q = \Pi_k u$. Computing the last part of the integral is no problem using the coefficients for $\Pi_k u$ and $\mathcal{I}_E$. The first part of the integral over u^2 might be more problematic, especially when we need it accurate. Again, it might be preferable to compute $\int_\Omega u^2 \, d\mathbf{x}$ and subtract $\int_\Omega (\Pi_k u)^2 \, d\mathbf{x}$. In both cases, we will loose some accuracy in the subtraction of the two integrals, just like with the approach in the previous section. The impact of this is hard to estimate beforehand.

4.5.3 Taylor polynomials

The third approach for the element vector is to use a Taylor polynomial approximation. In this case we would compute the coefficients r_i of the Taylor polynomial of u at the centroid of E. Computation then proceeds the same as in the second approach with as result

$$\int_E (u - u_h)^2 \, d\mathbf{x} \approx (\mathbf{r} - \mathbf{u}_h)^T \mathcal{I}_E (\mathbf{r} - \mathbf{u}_h). \tag{4.27}$$

Where $\mathbf{r}$ and $\mathbf{u}_h$ are the monomial coefficients for the Taylor polynomial of u and the solution u_h, respectively.

Compared to the previous approach, we keep the nice property that we have non-negative results. However, when we compare the truncated Taylor polynomial with the polynomial projection of u the latter introduces a smaller error in the approximation of u. Furthermore, unlike the polynomial projection approach, we do not know if the computed error using a Taylor polynomial over- or underestimates the real error.

4.5.4 Comparison

We have adapted the three alternative approaches for the element vector to allow computing the local error without element quadrature. As conclusion to this section we look whether these methods are of practical use.

To determine if they are better we first need to look at what accuracy is needed and what computational cost is allowed. As the main purpose for the error computation is verification of the correct behaviour of the implementation, we need a more accurate result than for the element vector. Otherwise the convergence rate might be dominated by the accuracy of the error computation and not by the error in the solution. Moreover, unlike the element vector, it may be possible to do some or all of the verification steps on a smaller mesh than for the actual full scale problem. Using a smaller mesh we can use methods that are more expensive per element, as there are fewer elements.

Additionally, the use of the method of manufactured solutions gives us some freedom in the choice of the exact solution u. Therefore, we might be able to choose a u such that it reduces the computational cost for the error computation. For example, we could choose u such that the functions G_j for the polynomial projection are easy to compute.

Still, we think that the triangulation and quadrature approach, if computationally feasible, will be the optimal approach. Its flexibility in allowable u, the better accuracy using higher order quadrature rules and its simplicity in implementation are all useful for the error computation.

When it is not possible to use the triangulation and quadrature approach due to computational cost or other reasons (e.g. the virtual DGVEM basis functions), we are not certain what is the best approach. We expect that the polynomial projection can give good approximations, but we are unsure of the cost. Using the Taylor polynomials to compute the error we are unsure about the accuracy of this approach.

4.6 Summary

In this chapter we looked at the ADG method. This method constructs basis functions by orthogonalizing the scaled monomials on each element, thereby allowing more general polygonal elements.

The orthogonalization requires the integration of polynomials over each element, for which traditionally an expensive triangulation and quadrature approach is used. We explained how the ideas from VEM can be used to exactly integrate the polynomials by constructing the matrix $\mathcal{I}_E$. With minimal work we were also able to compute the integrals needed in the element matrix A^E using $\mathcal{I}_E$.

We then continued by looking at replacing the integrals used in computing the element vector F^E and local error $\|u - u_h\|_E^2$. These are far more difficult, as both u and f are not necessarily polynomial. Therefore, we explored several different alternatives, each with its own strengths and weaknesses. We expect that the Taylor polynomial approach for f and the traditional triangulation and quadrature approach for u are the best choices, though this needs to be numerically verified and is of course dependent on the exact problem.

Chapter 5

DGVEM

With the ADG method discussed in the previous chapter we now turn to its competitor, the Discontinuous Galerkin version of the Virtual Element Method, presented in [16]. This combines the virtual basis functions of VEM with the discontinuous Galerkin approach of SIPG, thus allowing the more general polytopic element shapes from VEM with the various advantages of decoupling the basis functions of neighbouring elements.

In this chapter, we explain how this combined method works. We split this into three parts. First, we look at the relevant details of VEM that we did not discuss in Chapter 3. The second part looks into how to combine VEM with SIPG, because we need to make some slight modifications. The third part revisits the integration of the right hand side and error term, which is now essential as the virtual basis functions prevent the use of the triangulation and quadrature approach.

5.1 VEM basics

We already summarized the very core approach of VEM in our methods discussion (Chapter 3). That summary focused on how to compute the element matrix, but left out two important details: how to compute the projection Π^∇ and what stabilization term to use. For a complete implementation of VEM, we need, in addition to the element matrix, the element vector for which we will use the L^2-orthogonal projection.

5.1.1 Projection operator Π^∇

The most important part in VEM is the definition of the projection operator $\Pi^\nabla : V_k(E) \to \mathcal{P}_k(E)$. While defined in (3.4) using $\mathcal{A}^E$, it is for implementation more practical to rewrite the definition, leaving out the elementwise constant κ, to

$$\left(\nabla \Pi^\nabla v, \nabla p\right)_E = \left(\nabla v, \nabla p\right)_E \qquad \forall p \in \mathcal{P}_k(E). \tag{5.1}$$

This fixes $\Pi^\nabla v$ up to a constant, therefore we add the requirement

$$P_0 \Pi^\nabla v = P_0 v, \tag{5.2}$$

for some operator $P_0 : V_k(E) \to \mathbb{R}$. There are many such operators, but we use the choice of [13]:

$$P_0 v = \begin{cases} \frac{1}{n_{E,v}} \sum_{i=1}^{n_{E,v}} v(\mathbf{x}_i) & \text{if } k = 1, \\ \frac{1}{|E|} \int_E v \, \mathrm{d}\mathbf{x} & \text{if } k \geq 2. \end{cases} \tag{5.3}$$

This operator can easily be computed for the basis functions and, moreover, it works well with the L^2-orthogonal projection that we later define.

To actually use the operator, we need to be able to compute it for all the basis functions ϕ (we temporarily dropped the index). To do this, we use that $\Pi^\nabla \phi \in \mathcal{P}_k(E)$, so we can expand it in terms of the scaled monomials

$$\Pi^\nabla \phi = \sum_{i=1}^{n_{\mathcal{P},k}} s_i m_i,$$

where the s_i are the coefficients for monomials m_i. Plugging this into (5.1), we get

$$\sum_{i=1}^{n_{\mathcal{P},k}} s_i (\nabla m_i, \nabla m_j)_E = (\nabla \phi, \nabla m_j)_E \quad \text{for } j = 1 \cdots n_{\mathcal{P},k}.$$

As m_1 is a constant function we have that $\nabla m_1 = 0$. Hence, if we put this into a matrix system we would have zeros in both the first column and row. Therefore, we replace this row by the equation that follows from (5.2), and we get the system

$$\begin{pmatrix} P_0 m_1 & P_0 m_2 & \cdots & P_0 m_{n_{\mathcal{P},k}} \\ 0 & (\nabla m_2, \nabla m_2)_E & \cdots & \left(\nabla m_2, \nabla m_{n_{\mathcal{P},k}}\right)_E \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \left(\nabla m_{n_{\mathcal{P},k}}, \nabla m_2\right)_E & \cdots & \left(\nabla m_{n_{\mathcal{P},k}}, \nabla m_{n_{\mathcal{P},k}}\right)_E \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n_{\mathcal{P},k}} \end{pmatrix} = \begin{pmatrix} P_0 \phi \\ (\nabla m_2, \nabla \phi)_E \\ \vdots \\ \left(\nabla m_{n_{\mathcal{P},k}}, \nabla \phi\right)_E \end{pmatrix}. \tag{5.4}$$

While this defines the projection operator we still need to compute all the required matrix and vector entries.

The first row of the matrix depends on k. For $k = 1$, we can directly use the definition of P_0 and compute the monomials at the vertices. For $k = 2$, we need the average of the monomials m_j, which is equivalent to (m_j, m_0) as $m_0 = 1$. Thus the first row of the matrix is identical to the first row of $\mathcal{I}_E$.

We use a similar approach for the first element of the right hand side, where we use the definition of the basis functions. For $k = 1$, we get $P_0 \phi = \frac{1}{n_{E,v}}$, while for $k = 2$ it is zero, except when ϕ corresponds to the first moment degree of freedom, where it is 1.

The machinery for the remaining parts is already introduced. For the remaining bulk of the matrix, we can use the matrices from the previous chapter and write

$$(\nabla m_i, \nabla m_j)_E = \left[D_{x_1}^T \mathcal{I}_E D_{x_1} + D_{x_2}^T \mathcal{I}_E D_{x_2}\right]_{ij}.$$

For the innerproducts on the right hand side of (5.4), we observe that they match (3.3), for which we already described the process for the exact computation.

Combining all these computations, we can compute the coefficients s_i of the projection $\Pi^\nabla \phi$. Note, all these computations are exact, thus the resulting projection can be computed exactly using only the definition of ϕ in terms of the degrees of freedom.

5.1.2 Local element matrix

Using the projection operator, the element matrix was already defined in Chapter 3:

$$A_{ij}^E = \mathcal{A}^E\big(\Pi^\nabla \phi_i, \Pi^\nabla \phi_j\big) + \mathcal{S}^E\big(\phi_i - \Pi^\nabla \phi_i, \phi_j - \Pi^\nabla \phi_j\big), \tag{5.5}$$

where $\mathcal{S}^E$ is a stabilization term that we can compute using only the degrees of freedom on the element.

There are several choices for this stabilization term. The only requirement is that it should satisfy

$$c_0 \mathcal{A}^E(v,v) \le \mathcal{S}^E(v,v) \le c_1 \mathcal{A}^E(v,v) \qquad \forall v \in V_k(E) \quad \text{with } \Pi^\nabla v = 0,$$

for some constants c_0 and c_1 independent of E and h_E. We are using a slightly modified version of [12]:

$$\mathcal{S}^E(u,v) = \kappa_E \sum_{i=1}^{n_d} \chi_i(u)\chi_j(v),$$

where χ_i is the i-th degree of freedom, n_d is the number of degrees of freedom for element E. Furthermore, we added the constant κ_E, the value of κ on E, to match the bilinear form.

5.1.3 L^2-orthogonal projection

So far, we have defined the element matrix. The next logical step is the discuss the computation of the element vector. However, we first need to introduce the L^2-orthogonal projection onto the polynomials developed in [3]. Without this operator, we are limited to convergence of order k instead of the optimal $k+1$.

The L^2-orthogonal projection operator, denoted by $\Pi_k^0 : V_k(E) \to \mathcal{P}_k(E)$, is defined by

$$\int_E (\Pi_k^0 v) m \,\mathrm{d}\mathbf{x} = \int_E v m \,\mathrm{d}\mathbf{x} \qquad \forall m \in \mathcal{M}_k(E). \tag{5.6}$$

Note that Π_k^0 is a special case of Π_k, with as domain the VEM space, which allows it to be expressed as a matrix. For any $v \in V_k(E)$, we can compute most of the values on the right hand side from just the degrees of freedom. Unfortunately we can in general not do this for $m \in \mathcal{M}_{k-1}^\star(E) \cup \mathcal{M}_k^\star(E)$, as those moments are unknown.

Contrastingly, for $\Pi^\nabla v$ we know how to compute all the moments (it is a polynomial), but the moments do not need to align with v. Therefore, the idea from [3] is to take the moments for order up to $k-2$ from v and those of order $k-1$ and k from $\Pi^\nabla v$. To ensure that the moments of order $k-1$ and k of $\Pi^\nabla v$ coincide with those of v, we need to redefine our finite element space in a two step process. We first enlarge the original $V_k(E)$ to

$$\tilde{V}_k(E) = \big\{ v \in H^1(E) \;:\; v|_{\partial E} \in \mathcal{B}_k(\partial E),\ \Delta v \in \mathcal{P}_k(E) \big\}.$$

The larger space, with $\Delta v \in \mathcal{P}_k(E)$ instead of $\Delta v \in \mathcal{P}_k(E)$, allows us to impose the requirement that the moments of order $k-1$ and k of both $\Pi^\nabla v$ and v coincide, to get the subspace

$$W_k(E) = \Big\{ v \in \tilde{V}_k(E) \;:\; (v,q)_E = \big(\Pi^\nabla v, q\big)_E \; \forall q \in \mathcal{M}_{k-1}^\star(E) \cup \mathcal{M}_k^\star(E) \Big\}.$$

Comparing the new space with $V_k(E)$, we can still use the same degrees of freedom, we still have $\mathcal{P}_k(E) \subset W_k(E)$ and the projection $\Pi^\nabla : W_k(E) \to \mathcal{P}_k(E)$ can still be computed exactly. See for details [3]. Therefore we will replace the original definition of $V_k(E)$ with this new space $W_k(E)$.

Using this new space, we can now define the L^2-orthogonal projection $\Pi_k^0 : V_k(E) \to \mathcal{P}_k(E)$ using (5.6) and, more importantly, compute it. For the moments up to order $k-2$, we can still use the degrees of freedom. For the moments of order $k-1$ and k, we can now use the definition of $V_k(E) = W_k(E)$ to obtain

$$\int_E (\Pi_k^0 v) m \,\mathrm{d}\mathbf{x} = \int_E vm \,\mathrm{d}\mathbf{x} = \int_E (\Pi^\nabla v) m \,\mathrm{d}\mathbf{x} \qquad \forall m \in \mathcal{M}_{k-1}^\star(E) \cup \mathcal{M}_k^\star(E).$$

The actual matrix computation is relatively easy and proceeds along the same lines as with the Π^∇ operator. Moreover, due to the choice of P_0 in (5.3), we have $\Pi^\nabla = \Pi_k^0$ for $k \leq 2$ [3], which we can use either as optimization or for verification of our computations.

This new function space might look like it requires some changes to the numerical implementation, but that is not the case. For the computations of Π^∇ and A^E we have only used that the functions are polynomial on all the edges and that the basis functions associated with the degrees of freedom are orthogonal. While the basis functions might have changed, that will not affect our computation as they remain virtual. Hence, no changes to the code are needed when using $W_k(E)$ as $V_k(E)$ instead of the original $V_k(E)$ from (3.2).

5.1.4 Element vector

Now that we have covered the L^2-orthogonal projection, we can discuss the computation of the element vector

$$F_i^E = \int_E \phi_i f \,\mathrm{d}\mathbf{x}. \tag{5.7}$$

As the ϕ_i are virtual, we can not compute this integral directly.

If f would be a polynomial of order at most $k-2$, we would be able to compute the integral. The approach in this case is to expand f in terms of the scaled monomials, and computing the resulting integrals of the form $\int \phi_i m_j \,\mathrm{d}\mathbf{x}$ using the moment degrees of freedom. Therefore [12] alternatively defines:

$$F_i^E = \int_E \phi_i (P_{k-2} f) \,\mathrm{d}\mathbf{x}, \tag{5.8}$$

where $P_{k-2} f$ is a projection of f onto the space $\mathcal{P}_{k-2}(E)$.

Using such a low order projection for f has as disadvantage that the convergence of the method is limited to h^k [12]. We can obtain a better result by using

$$F_i^E = \int_E \phi_i (P_{k-1} f) \,\mathrm{d}\mathbf{x},$$

or even

$$F_i^E = \int_E \phi_i (P_k f) \,\mathrm{d}\mathbf{x}. \tag{5.9}$$

Both give the optimal convergence rate h^{k+1} [3]. The integral (5.9) can be computed using the L^2-orthogonal projection operator using

$$F_i^E = \int_E \phi_i (P_k f)\,\mathrm{d}\mathbf{x} = \int_E (\Pi_k^0 \phi_i)(P_k f)\,\mathrm{d}\mathbf{x}, \tag{5.10}$$

in combination with the inner product matrix $\mathcal{I}_E$. This shows why [12] uses (5.8), the steps to compute the projection operator Π_k^0, required in (5.10), were published later in [3].

This also motivates why [12] does not use this approach, the changes for the L^2-orthogonal projection operator were published later.

Just as with ADG the question is how to define P_k so that we can compute $P_k f$ efficiently. We will discuss several options in Section 5.3.

5.2 SIPG and VEM

Now that we have a basic understanding of VEM, we can combine it with SIPG. Comparing both VEM and SIPG to the continuous Galerkin method, we see that both have modified the bilinear form $\mathcal{A}^E$ and the linear form $\mathcal{F}^E$. Therefore, there is some work needed in combining them both. We follow the work of [16], which did this for a Poisson problem with zero Dirichlet boundary conditions (i.e. $g = 0$).

We start by looking at the bilinear form. The modification for SIPG is the addition of fluxes between the elements and the integrals over the edges of the elements (2.14). VEM modifies the element integral itself so that it can be computed without explicit basis functions. Naively combining them we would get

$$\begin{aligned}
A_{ij}^E &= \kappa \int_E \nabla(\Pi^\nabla \phi_i) \cdot \nabla(\Pi^\nabla \phi_j) + \mathcal{S}^E\left(\phi_i - \Pi^\nabla \phi_i, \phi_j - \Pi^\nabla \phi_j\right) \mathrm{d}\mathbf{x}, \\
A_{ij}^e &= \int_e -\{\!\{\kappa \nabla \phi_i\}\!\}_{\beta_e} [\![\phi_j]\!] - \{\!\{\kappa \nabla \phi_j\}\!\}_{\beta_e} [\![\phi_i]\!] + \gamma_e [\![\phi_i]\!][\![\phi_j]\!] \,\mathrm{d}S.
\end{aligned}$$

In combining the VEM and DG methods, we have introduced the terms $\{\!\{\kappa\nabla\phi_i\}\!\}_{\beta_e}$ and $\{\!\{\kappa\nabla\phi_j\}\!\}_{\beta_e}$. Unfortunately, we can not compute $\nabla\phi$ using only the degrees of freedom. As a solution, we project ϕ using Π^∇, in order to be able to compute the gradient, this gives

$$A_{ij}^e = \int_e -\{\!\{\kappa \nabla \Pi^\nabla \phi_i\}\!\}_{\beta_e} [\![\phi_j]\!] - \{\!\{\kappa \nabla \Pi^\nabla \phi_j\}\!\}_{\beta_e} [\![\phi_i]\!] + \gamma_e [\![\phi_i]\!][\![\phi_j]\!] \,\mathrm{d}S.$$

Note that, with a slight abuse of the notation, we extended Π^∇ from a operator defined on a single element, to a global one coinciding with the local Π^∇ for the element.

By the same reasoning, we can combine the element F^E and edge F^e vectors of SIPG and VEM to get

$$\begin{aligned}
F_i^E &= \int_E \phi_i (P_k f)\,\mathrm{d}\mathbf{x}, \\
F_i^e &= \int_e g\left[-\mathbf{n} \cdot \kappa \nabla \Pi^\nabla \phi_i + \gamma_e \phi_i\right] \mathrm{d}S.
\end{aligned}$$

Note that for F^E we choose to use (5.9), though we could have used (5.8) or even different approximations. The important change is in the edge vector F^e, where we again need the polynomial projection to be able to compute the gradient.

5.3 Efficient right hand side

In the introduction of how VEM handles the computation of the element vector, we introduced the projection operator P_k. This operator projects the right hand side onto the polynomials, thereby allowing the computation of the element vector using only the degrees of freedom. The question then remains of how should we define P_k, preferably so that we can compute $P_k f$ relatively efficiently.

In Section 4.4 we faced a similar problem when computing the element vector for ADG. There we proposed four different solutions:

1. direct computation of using a conversion to a boundary integral,
2. using the boundary conversion approach to compute the L^2-orthogonal projection of f onto the polynomials,
3. approximate it using a Taylor polynomial,
4. triangulating the element and using standard quadrature.

Options two and three are clearly candidates for a projection P_k.

The fourth options is not possible using VEM. We can not compute the value of the basis functions ϕ_i inside the element, which makes the use of a quadrature rule on a triangulation impossible.

The first option is also not practical. There we need to find functions G_i, such that $\nabla \cdot G_i = \phi_i f$ and then use

$$\int_E \phi_i f \,\mathrm{d}\mathbf{x} = \int_{\partial E} G \cdot \mathbf{n} \,\mathrm{d}S.$$

As the basis functions ϕ_i are unknown and in general non-polynomial, we do not expect to be able to find such G_i. This is in contrast to the second option, where we would need integrals of the form $\int_E m_i f \,\mathrm{d}\mathbf{x}$.

5.3.1 VEM interpolant

For VEM we have a different approach, using that $V_k(E)$ contains more functions than only polynomials. Hence, we consider approximating f with some $f_I \in V_k(E)$ and use that for the computation of F^E.

To define f_I, we use that the degrees of freedom $(\chi_i(\cdot))$ implicitly define a natural interpolation. The interpolant is defined as the function $f_I \in V_k(E)$ that satisfies

$$\chi_i(f) = \chi_i(f_I), \qquad i = 1, 2, \ldots, n_d.$$

Where the moment degrees of freedom can be computed by using triangulation and quadrature, or by using the divergence theorem and integration over the boundary of E. Using the interpolant f_I as approximation of f in F^E we get

$$\mathcal{F}_i^E = \int_E \phi_i f \, \mathrm{d}\mathbf{x} \approx \int_E \phi_i f_I \, \mathrm{d}\mathbf{x}.$$

To see if we can compute this, we observe that $f_I \in V_k(E)$ and thus we can express it as a vector with the values of its degrees of freedom $\mathbf{f}_I$. Then the integral is equivalent to

$$\int_E \phi_i f_I = \sum_{j=1}^{n_d} \mathcal{M}^E(\phi_i, \phi_j)\mathbf{f}_{I,i},$$

with

$$\mathcal{M}^E(\phi_i, \phi_j) = \int_E \phi_i \phi_j \, \mathrm{d}\mathbf{x}.$$

Using the standard polynomial basis functions the computation of these integrals is no problem resulting in the mass matrix. For VEM this is, of course, a bit more complicated and we need the approximation from [3]. Using the same steps as with the stiffness matrix they propose

$$\mathcal{M}^E(\phi_i, \phi_j) \approx M_{ij} = \mathcal{M}^E(\Pi_k^0 \phi_i, \Pi_k^0 \phi_j) + \mathcal{S}_0^E(\phi_i - \Pi_k^0 \phi_i, \phi_j - \Pi_k^0 \phi_j).$$

Where now $\mathcal{S}_0^E$ is a suitable symmetric bilinear stabilization term similar to $\mathcal{S}^E$ for the stiffness matrix. Using this computable approximation gives

$$F^E = M\mathbf{f}_I. \tag{5.11}$$

Thus, we can use this alternative approach for the computation of the right hand side.

5.3.2 Comparison

With these three approaches (Polynomial projection, Taylor polynomial, VEM interpolation) we have the question which one to use. For ADG we already argued that the Taylor polynomial is probably the least expensive and exact enough for the right hand side computation. The question is thus how does the VEM interpolation compare to both methods.

For this we have to consider the cost. To find the VEM projection f_I we need to compute moments of order up to $k-2$ and the value at the boundary degrees of freedom. Typically this is cheaper than the computation of all the k moments of f for the polynomial projection, as moments are more expensive to compute than the value of f. In addition, we have to compute the mass matrix approximation M, which is not a cheap operation, as it requires inverting $\mathcal{I}_E$.

When considering the gains in accuracy, we only know that f_I is a function in $V_k(E)$. This space contains $\mathcal{P}_k(E)$, which contains the polynomial projection of f, thereby allowing for a more accurate approximation. However, we do not know if f_I is actually a better approximation, and whether this accuracy is lost when using the approximated mass matrix.

Comparing VEM interpolation to the Taylor polynomial approach we can be brief. As VEM interpolation requires the computation of both moments of f and the approximated mass matrix it will be much more expensive. Therefore, we expect that the Taylor polynomial, assuming sufficiently accurate, is the better approach.

Comparing VEM interpolation to polynomial projection, we expect that the use of the VEM interpolant might be computationally beneficial for problems where we already computed the mass matrix. For such problems, we do not incur the extra penalty of computing the mass matrix, while we reduce the number of moments of f that we need to compute.

5.4 Error computation

Just as with ADG, we also need to look at the error computation. For VEM this is even a bigger problem, as not only do we have the product between a virtual solution u_h and the unknown function u, we also have the problem of integrating the product u_h^2. We investigated several options.

5.4.1 Polynomial projection

The first option is to use a polynomial projection, in this case of the VEM solution. By computing $\Pi_k^0 u_h$ we get a polynomial approximation of the VEM solution, which we can use for the error computation.

Here we can choose two directions. The simplest is to triangulate the element and then perform a quadrature to compute

$$\int_E \left(u - \Pi_k^0 u_h\right)^2 \mathrm{d}\mathbf{x}. \tag{5.12}$$

An approach that is very straightforward in implementation, but not very cheap to compute.

Alternatively, we can also project u to $\mathcal{P}_k(E)$ using P_k. Using this approach, the integration becomes almost trivial as both $P_k u$ and $\Pi_k^0 u$ can be expressed as monomial coefficients. Let $\mathbf{u}_d$ be the vector with the differences in the coefficients for the scaled monomials, and we have

$$\int_E \left(P_k u - \Pi_k^0 u\right)^2 \mathrm{d}\mathbf{x} = \mathbf{u}_d^T \mathcal{I}_E \mathbf{u}_d. \tag{5.13}$$

This can be faster (depending on the speed of computing the moments of u), but throws away even more detail of the solution. Projecting u to a higher polynomial space is of course possible, but increases both the cost of the projection and the cost of computing the larger $\mathcal{I}_E$.

In the extreme case, we could forgo all attempts at creating a good error approximation and use as projection $P_k u$ the local Taylor polynomial of u. Though, at that point one can certainly argue that we are stacking so many approximations, that we only get a rough idea of the error size instead of an actual approximation.

5.4.2 VEM interpolation

A better attempt may be found using the VEM interpolant u_I, constructed using the same process as the VEM interpolation f_I of f. Using this interpolant, we can compute $u_I - u_h \in V_k(E)$ exactly. Moreover, let $\mathbf{u}_d$ be the vector with the degrees of freedom of this difference. Then we can compute the L^2-norm of this difference using the approximate mass matrix M:

$$\int_E (u - u_h)^2 \,\mathrm{d}\mathbf{x} \approx \mathbf{u}_d^T M \mathbf{u}_d.$$

Using this approach, we are again introducing approximations by first interpolating u to get u_I and then using the approximate mass matrix. However, unlike the polynomial projection approaches, we are not projecting the solution u_h from $V_k(E)$ to a lower dimensional space like $(\mathcal{P}_k(E))$. This prevents the loss of accuracy in projecting the solution to $\mathcal{P}_k(E)$. How much accuracy we loose in creating u_I and using M is unfortunately not really known.

5.4.3 Solution reconstruction

For the third option, we go against the idea of VEM not to compute the basis functions. Using the definition of $V_k(E)$ and the computed degrees of freedom we can reconstruct the actual solution u_h. This solution can then be used to compute the difference $u - u_h$ and compute the error. As the basis functions of VEM are not designed to be computed, this will be computationally expensive and thus not useful for actual large scale problems.

Looking at the requirements for $V_k(E)$ in (5.1.3), we see that our reconstructed solution $\bar{u}_h$ should satisfy:

- $\Delta \bar{u}_h = p$ for some $p \in \mathcal{P}_k(E)$ inside E,
- $\bar{u}_h|_{\partial E} \in \mathcal{B}_k(\partial E)$, fixing the value on the boundary of E,
- the moments of order $k-1$ and k for $\bar{u}_h$ and $\Pi^\nabla \bar{u}_h$ should coincide: $\int_E \bar{u}_h m \,\mathrm{d}\mathbf{x} = \int_E \left(\Pi^\nabla \bar{u}_h\right) m \,\mathrm{d}\mathbf{x}$ for $m \in \mathcal{M}^\star_{k-1}(E) \cup \mathcal{M}^\star_k(E)$.

Moreover, as $\bar{u}_h$ is the reconstruction of u_h, we also expect that the degrees of freedom match, thus $\chi_i(\bar{u}_h) = \chi_i(u_h)$.

As we do not know the actual polynomial p in the first constraint, we will need to solve the Poisson equation several times. Hence, we create a mesh of E and our reconstruction $\bar{u}_h$ will be an approximation to u_h from a suitable finite element space.

For the actual computation, we start with the second constraint and construct $u_{h,\partial} \in \mathcal{B}_k(\partial E)$ using the boundary degrees of freedom. This function is then used as boundary condition and solve

$$\begin{cases} \Delta u_h^0 = 0 & \text{in } E, \\ u_h^0 = u_{h,\partial} & \text{on } \partial E. \end{cases} \tag{5.14}$$

The solution u_h^0 obviously satisfies the first two constraints that we placed on $\bar{u}_h$ and has matching boundary degrees of freedom. However, the last constraint is not satisfied, nor is the equality of the moment degrees of freedom. To satisfy both we need that $\Delta u_h^0 \in \mathcal{P}_k(E)$. We do not know the coefficients of this polynomial, but we can solve

$$\begin{cases} \Delta u_h^i = m_i & \text{in } E, \\ u_h^i = 0 & \text{on } \partial E, \end{cases} \tag{5.15}$$

where the m_i are the usual scaled monomials with $i = 1, 2, \ldots, n_{\mathcal{P},k}$. Using these solutions we define our reconstruction of the solution as

$$\bar{u}_h = u_h^0 + \sum_{i=1}^{n_{\mathcal{P},k}} \alpha_i u_h^i,$$

where the α_i are constants that still have to be determined.

To find the constants α_i we will reuse the definition of Π_k^0 for u_h (5.6):

$$\int_E (\Pi_k^0 u_h) m \,\mathrm{d}\mathbf{x} = \int_E u_h m \,\mathrm{d}\mathbf{x}, \qquad \forall m \in \mathcal{M}_k(E).$$

The values on the left hand side are computable using the actual degrees of freedom on u_h and the already developed machinery of Π_k^0 and $\mathcal{I}_E$. On the right hand side, we plug in the approximation $\bar{u}_h$ to get

$$\int_E (\Pi_k^0 u_h) m \,\mathrm{d}\mathbf{x} = \int_E u_h^0 m \,\mathrm{d}\mathbf{x} + \sum_{i=1}^{n_{\mathcal{P},k}} \alpha_i \int_E u_h^i m \,\mathrm{d}\mathbf{x}.$$

The actual values on the right hand side can be computed using standard quadrature on the mesh of E used for computing $\bar{u}_h$. Plugging in all the monomials $m_i \in \mathcal{M}_k(E)$ as m, we get $n_{\mathcal{P},k}$ equations, which is sufficient to find the $n_{\mathcal{P},k}$ constants α_i, thereby fully determining our reconstruction $\bar{u}_h$.

After all this work we can finally use the result to compute the error $\|u - u_h\|_E^2 \approx \|u - \bar{u}_h\|_E^2$. Again, we compute this integral using the mesh for E used in the construction of $\bar{u}_h$ along with standard quadrature rules.

5.4.4 Comparison

From all these options we see that the error computations for VEM can only be estimates. We can get arbitrarily close when reconstructing the solution, but at the hefty price of solving $n_{\mathcal{P},k} + 1$ Poisson equations per element. The polynomial and VEM interpolant are faster, but only give a rough approximation of the error.

Having to solve that many PDE's per element reduces the practical use of the solution reconstruction method. Therefore we do not expect it to be useful for any larger scale error computations. For that we can still use VEM interpolation and polynomial projection, though these methods will introduce some inaccuracy.

Comparison of using the VEM interpolant or the polynomial projection approach is almost pure speculation. The projection of u_h to the polynomials for computing its error, also introduces an error. Though using the VEM interpolant can be better, but introduces errors due to the use of an approximate mass matrix.

5.5 Summary

In this chapter we looked at DGVEM. This method uses the non-polynomial, but conforming basis functions of VEM, combined with the discontinuous Galerking approach of SIPG. By using these VEM virtual basis functions, it also inherits the complications that stem from not computing them. The construction of the element matrix and element vector require two projections of the basis functions, $\Pi^\nabla \phi$ and $\Pi_k^0 \phi$, that need to be computed. Moreover, the non-polynomial functions f and u in the element vector and error computations, respectively, cause more difficulty than with ADG.

For the element vector we can not use the triangulation and quadrature approach, but the polynomial projection and Taylor polynomial options remain possible, with the addition of the VEM interpolant. Just as with ADG, we expect that the Taylor polynomial to be the computationally cheap, but sufficiently accurate option.

For the error computation, we, unfortunately, have less appealing choices. We can project the solution back to polynomials, loosing information and accuracy of the solution. Alternatively, we can interpolate the solution u to VEM space, but have to use an expensive and only approximate mass matrix. The third alternative of reconstruction of u_h is can be very accurate but hideously expensive, as it requires solving many Poisson problems on each of the elements in the mesh. This need to solve more Poisson problems on smaller meshes to compute the error, defeats the purpose of using DGVEM to solve our original Poisson problem on a large mesh. Therefore, it is unusable for practical problems, but could be useful for more theoretical purposes.

Chapter 6

Numerical results

With the theoretical side of ADG and DGVEM covered, we can now turn to the numerical results. We start with standard convergence results to the verify the correct behaviour of the method. Certain that our methods work as intended, we then continue by looking at the different element vector and error computation possibilities for both ADG and DGVEM. We finish with the most important part of the numerical results, a comparison between ADG and DGVEM.

For these numerical discussions we will need to fix the accuracy order k for the methods. Therefore, we will write ADGk and DGVEMk to denote the ADG and DGVEM methods using a specific value of k, e.g. ADG1 is ADG with $k = 1$. Furthermore, unless stated otherwise we will assume $\gamma = 20$ as stabilization parameter and $\kappa = 1$.

6.1 Verification

To verify that both ADG and DGVEM work as expected, we test them using two manufactured solutions. For both methods we solve our generalized Poisson model problem (2.1), as domain Ω we use the unit square, with Dirichlet boundary conditions around the full boundary (i.e. $\Gamma_D = \partial\Omega$). On this domain we use a mesh of regular triangles, see Figure 6.1.

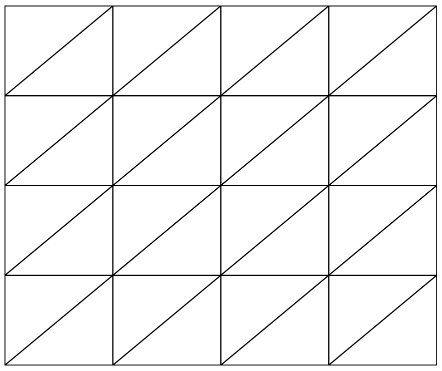
The first verification is done using the manufactured solution

$$u(x_1, x_2) = \sin(2\pi x_1)\sin(2\pi x_2), \tag{6.1}$$

with as consequence

$$\begin{cases} f(x_1, x_2) = 8\pi^2 u(x_1, x_2) & \text{in } \Omega, \\ g_D = 0 & \text{on } \partial\Omega. \end{cases} \tag{6.2}$$

We computed the solution using the appropriate f for four methods. As baseline we use SIPG in physical coordinate space from Chapter 2. For ADG and SIPG, we use the polynomial projection (4.23), with (4.18) for required moments, for the element vector and quadrature for the error computation. For DGVEM, we use again the polynomial projection (5.9), with (4.22) and (4.18), for the element vector, while the error is computed by projecting the solution to


(a) Smallest mesh

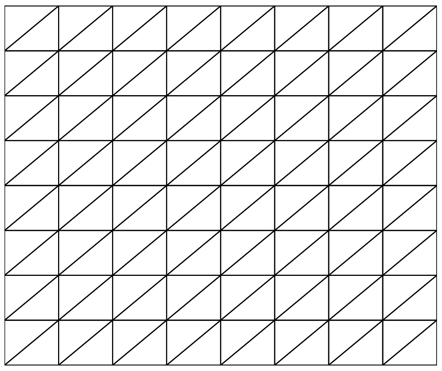
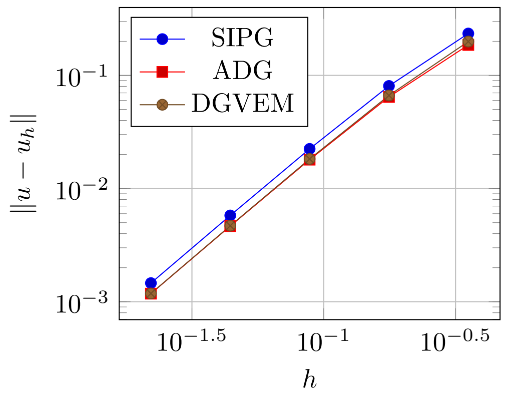
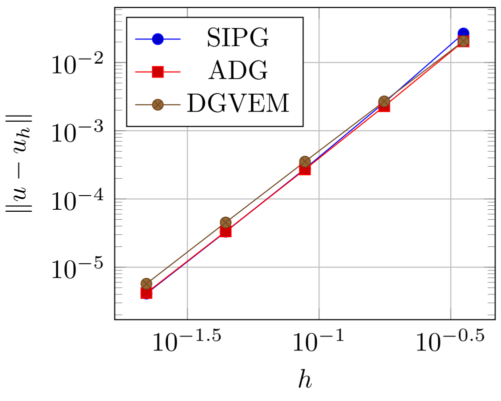

(b) First refinement

Figure 6.1: First two meshes used for the first convergence test.



(a) $k = 1$



(b) $k = 2$

Figure 6.2: Convergence for the first testcase using $\gamma = 20$. Note that for $k = 1$ the lines of ADG and DGVEM overlap, while for $k = 2$ the lines of SIPG and ADG overlap.

Table 6.1: Error norm $\|u - u_h\|$ for the second test case, using $\gamma = 20$ and the mesh of Figure 6.1b. The column marked with D is with Dirichlet boundary conditions on the whole boundary, while the DN uses Neumann boundary conditions on the boundary part with $x_2 = 0, 1$.

Method	Order	L^2-Error (D)	L^2-Error (DN)
SIPG	1	4.5×10^{-15}	3.3×10^{-15}
	2	9.5×10^{-15}	1.3×10^{-14}
ADG	1	4.8×10^{-15}	1.4×10^{-14}
	2	6.4×10^{-15}	2.3×10^{-14}
DGVEM	1	1.8×10^{-14}	5.1×10^{-14}
	2	2.6×10^{-14}	5.6×10^{-14}

polynomials and using quadrature (5.12). We can see in Figure 6.2 that all three methods converge at the expected rate h^{k+1}.

The second verification is done to test with discontinuous κ, specifically, one of the form

$$\kappa = \begin{cases} \frac{1}{2} & \text{for } x_1 \leq \frac{1}{2}, \\ 10 & \text{for } x_1 > \frac{1}{2}. \end{cases}$$

Additionally, we choose as solution,

$$u = \begin{cases} 2x_1 \frac{\kappa_2}{\kappa_1 + \kappa_2} & \text{for } x_1 \leq \frac{1}{2}, \\ 1 - 2x_1 \frac{\kappa_1}{\kappa_1 + \kappa_2} & \text{for } x_1 > \frac{1}{2}, \end{cases}$$

which gives as problem

$$\begin{cases} f = 0 & \text{in } \Omega g_D = u|_{\partial\Omega} \,. \end{cases}$$

Note that this solution is piecewise linear. Both ADG and DGVEM can theoretically exactly reproduce this solution. In Table 6.1, we see that the error is, as expected, near machine precision.

6.2 ADG

Now that we know that both methods work as intended, we can compare the alternatives for the element vector. We start with ADG, and the alternatives that were introduced in Chapter 4.

To tests these, we need one or more polygonal meshes. We choose to create meshes by using a proto-tile to tile the plane and cut a square section out of it. As result several tiles at the boundary are cut off, creating possibly elements that are small or that have large aspect ratio. Creating more refined meshes is easy, we can simply tile our domain with a smaller proto-tile. Specifically, for our test cases we create smaller meshes by scaling each of the proto-tiles' edges by a factor $\frac{1}{2}$.

The resulting meshes that are generated (at the largest level), are shown in the first column of Figure 6.3. As can be seen in this figure, we use five proto-tiles: triangle, hexagon, arrow, brick and boat. The triangle, as it is traditionally used in finite element methods. The hexagon, arrow and brick shapes because they all have six vertices, which ensures that ADG2 and DGVEM1 both have six degrees of freedom per element. With both convex (hexagon) and non-convex (arrow) elements and additionally one with hanging nodes (brick). Lastly, the boat shape has two reflex angles, creating a possibly ill-conditioned element.

6.2.1 Norm

As first alternatives for ADG, we consider the norm computation. We consider four alternatives for the computation of $\|u - u_h\|_E$:

1. Traditional quadrature on the possibly triangulated element. To increase the accuracy, we require that the quadrature is accurate for polynomials up to order $2k + 4$.
2. The polynomial projection to k-th order polynomials (4.26). The projection $\Pi_k u$ is computed using (4.22) with $l = k$ and (4.18) to compute the moments. In the last computation we used a line quadrature rule accurate for polynomials of order $2k + 3$.
3. A local Taylor polynomial approximation of u of order k. Then using (4.27) for computing the actual error.
4. The same Taylor polynomial approach, but with a polynomial of order $k + 2$. Note that this requires the computation of a larger $\mathcal{I}_E$.

In addition to these, we reuse the sine test case from (6.2). The element vector is computed by using triangulation and quadrature, with a quadrature rule accurate for polynomials up to order $2k$.

The results of using these approaches with ADG2 are shown in Figure 6.3. We observe that for all the tile shapes the accuracy is at the expected rate of h^3 for the higher refinement levels. Furthermore, comparing the methods, we see several trends that are independent of the tile shape.

Using the triangulation and quadrature approach as reference, we see the following behaviour. The Taylor polynomial of order k reports a significantly larger error. The Taylor polynomial of order k+2 reports an almost identical error as the triangulation and quadrature approach. While using the polynomial projection of u the reported error is smaller, just as we expected.

Comparing the runtimes, we see that polynomial projection is about twice as computationally expensive as the triangulation and quadrature approach. While the Taylor polynomial of order k is far cheaper, and the $k + 2$ order one is much more expensive. The latter is due to the computation of $\mathcal{I}_E$ for order $k + 2 = 4$. Note, that we recompute the complete $\mathcal{I}_E$ for this instead of reusing the already known $\mathcal{I}_E$. While this could be optimized, we do not expect a speed gain of more than a factor two.

To verify that these trends are not dependant on the choice of $k = 2$, we also computed some of the same test cases using ADG1 and ADG4. In Figure 6.4 we show the results for the boat tile mesh. The general trends that we saw for the ADG2 tests are also visible here.

In addition to these trends, we see some very large errors for ADG1, with its convergence speed increasing with the smaller and smaller elements. Visually inspecting the solution at the two most refined meshes, we see that solutions approximates the theoretical one, though with large discontinuities between the elements. Moreover, looking at the amplitude of the solution, we observe a value of 0.67 for level three and 0.82 for level four instead of the expected 1. For all

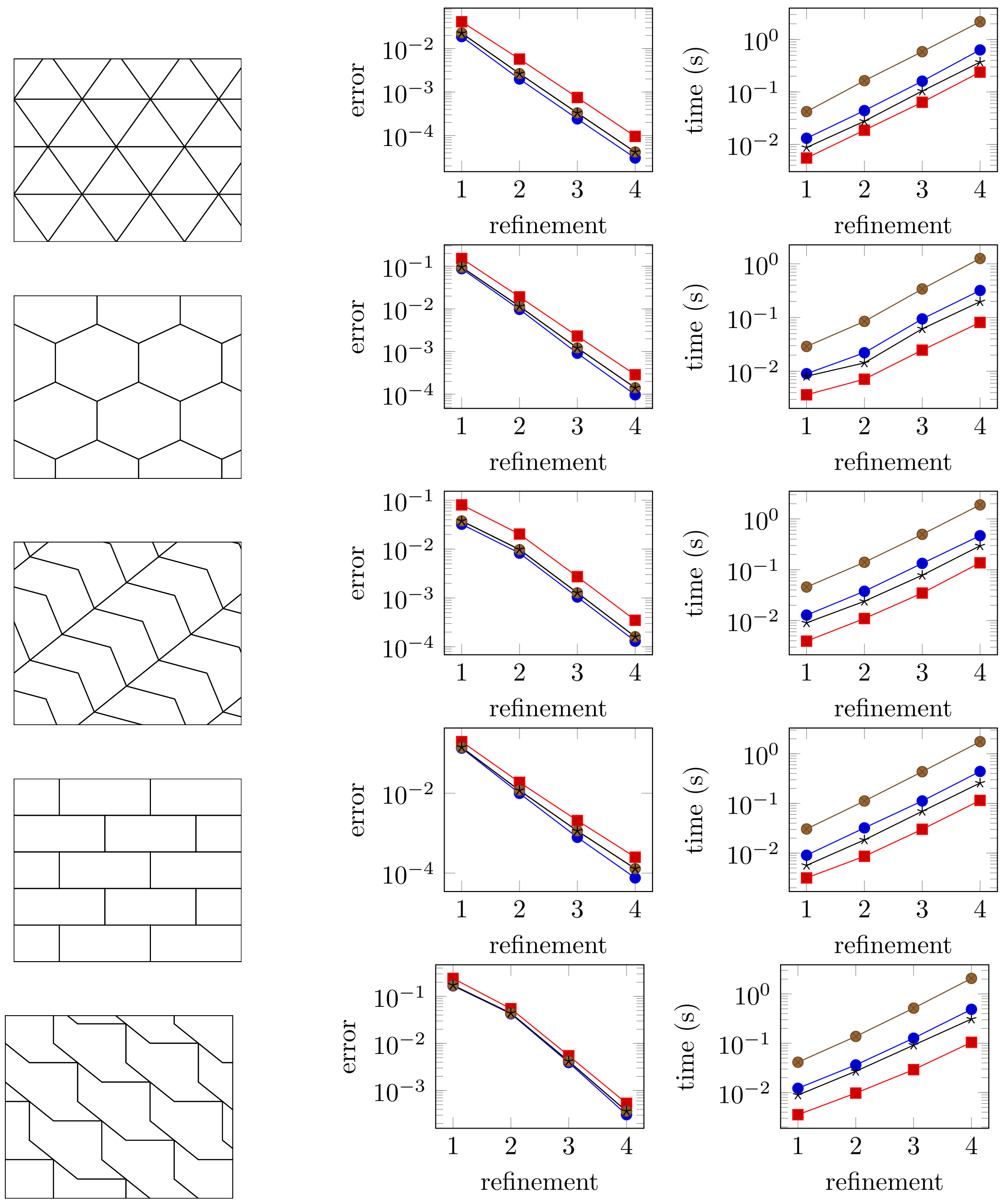


Figure 6.3: Error norm comparison for ADG2, comparing polynomial projection of order k (blue), Taylor polynomial of order k (red) and $k+2$ (brown) and the triangulation and quadrature of order $k+2$ (black). The first entry in each row of the figure is the first computational mesh, used to compute the solution u_h. In the second column the reported error $\|u-u_h\|_\Omega$ by these methods. The third entry in each row is the runtime of the norm computation. The horizontal axis corresponds to the mesh refinement, where each level decreases the edge length of the tile by a factor $\frac{1}{2}$. Note that lines for the order $k+2$ Taylor polynomial (brown) and the triangulation and quadrature (black) overlap in the error diagram.

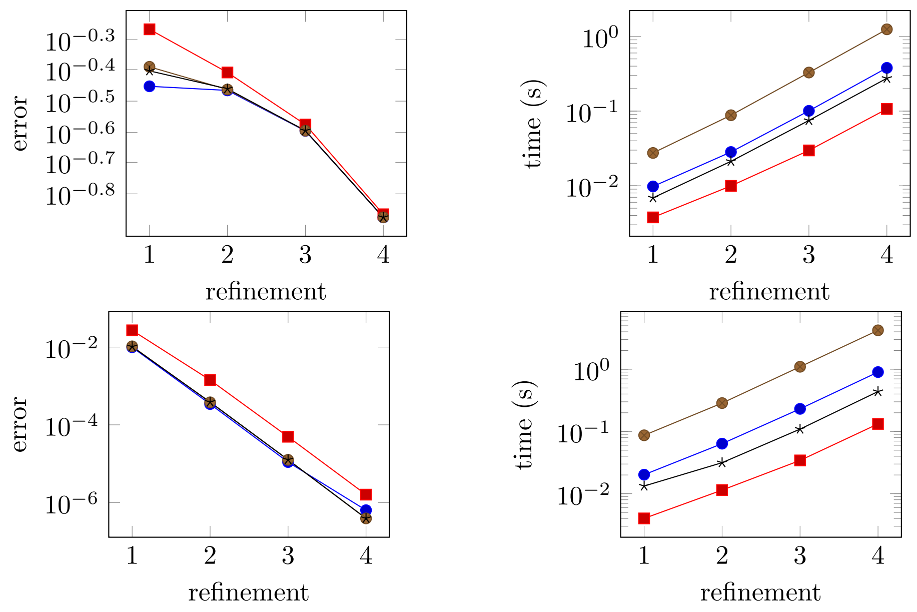


Figure 6.4: Similar error figures to Figure 6.3, but now using ADG1 (top) and ADG4 (bottom) on the boat tile mesh. We use the same colors as in Figure 6.3, with polynomial projection in blue, Taylor polynomial of order k and $k + 2$ in red and brown, respectively, and quadrature of order $k + 2$ in black. Again, we have that in error figures several lines overlap. In this case all three lines for the triangulation and quadrature (black), the Taylor polynomial of order $k + 2$ (brown) and the polynomial projection.

the other meshes, except the triangle one, a similar, but less extreme error in the amplitude is observed for ADG1.

From these tests, we conclude that the triangulation and quadrature approach is the most optimal approach for ADG. It does give accurate results, while being faster than the polynomial projection and higher order Taylor polynomial approaches. This is even without considering the better flexibility with respect to the choice of f and the required accuracy.

6.2.2 RHS

Having determined that the traditional triangulation and quadrature approach is indeed the most optimal for ADG, we turn our attention to the element vector computation. Where we consider consider three approaches

- Triangulating the element and then performing quadrature. We are using a quadrature rule accurate for polynomials of order up to $2k$.
- Polynomial projection of f, using (4.23) for $l = k$ in combination with (4.18) for the moment computation. In the moment computation, we again use a line quadrature that is accurate for polynomials of order up to $2k + 3$.
- An Taylor polynomial approximation of order k of f, using (4.24) for the computation of the element vector.

Table 6.2: Comparison of the time used by the triangulation and quadrature (TQ), projection and Taylor polynomial expansion approaches to the element vector computations on the triangle tile mesh. The runtime column is the runtime of the ADG2 with a zero element vector, while the relative (rel.) columns are the relative runtime with respect to the triangulation and quadrature approach.

level	Runtime (s)	TQ time(s)	Projection		Taylor	
			time (s)	rel.	time (s)	rel.
1	0.04	0.0025	0.0096	3.9	0.0026	1.1
2	0.12	0.0092	0.036	3.9	0.0098	1.1
3	0.44	0.031	0.13	4.4	0.038	1.2
4	1.6	0.1	0.48	4.8	0.14	1.4

Table 6.3: Similar to Table 6.2, but for the meshes with boat shaped elements. Note that the runtime of the alternative methods is similar to that on triangles, while for the triangulation and quadrature the runtime is more than doubled.

level	Runtime (s)	TQ time(s)	Projection		Taylor	
			time (s)	rel.	time (s)	rel.
1	0.036	0.0049	0.0096	2	0.0016	0.32
2	0.1	0.018	0.03	1.7	0.005	0.29
3	0.37	0.064	0.11	1.8	0.017	0.26
4	1.4	0.25	0.44	1.7	0.062	0.25

We use each of these approaches in ADG2 to solve the sine test problem. During the computation we measure the time that the element vector computation takes, while afterwards we compute the error in the result using triangulation and quadrature (exact for polynomials of order $2k + 2$).

Doing this on all 20 meshes (5 tiles, 4 levels) of the norm comparison, we do not note significant changes in the accuracy compared to triangulation and quadrature. With both the Taylor polynomial and the polynomial projection giving errors of about 0.8 to 1 times that of the triangulation and quadrature approach, for almost all meshes.

Therefore, we look at the computational cost, specifically the runtime (wall time) of computing the element vector using these approaches. We computed them on all the meshes, but list them here for both the triangular mesh in Table 6.2 and for the boat mesh in Table 6.3.

For triangular meshes, we only need to triangulate a few boundary elements, therefore the triangulation and quadrature approach is relatively cheap. As result, we see that the polynomial projection approach is far more expensive while the, Taylor polynomial is only slightly more expensive.

To triangulate the boat tile, we need six triangles, hence using triangulation and quadrature on this mesh is much more expensive. As result, we see that the Taylor polynomial approach is relatively faster. The polynomial projection is, even with the difference, still slower than the triangulation and quadrature approach.

In conclusion, we see that Taylor polynomial approach can be a good alternative to the triangulation and quadrature approach. It allows for faster element vector computation, while retaining approximately the same accuracy.

6.3 DGVEM

Having considered the alternatives for ADG, we now look at the options for DGVEM. We take the same approach using the sine test case with the same meshes.

6.3.1 Norms

Starting with the norms, we have three approaches that we consider:

- Projecting both numerical and actual solution to the polynomials of order k, then compute the error using (5.13). For the projection of the actual solution, we take the same approach as we did with ADG in the previous section.
- Using a VEM interpolation of the solution u and computing the error with (5.4.2), where we compute the required moments of u using (4.18).
- Reconstructing the solution using the PDE-toolbox in MATLAB and the approach of Section 5.4.3, i.e. compute a finite element approximation of the VEM solution. Internally the PDE-toolbox uses a first order continuous Galerkin method. For the computation of $\|u - \bar{u}_h\|_E$ we use a quadrature rule of order $k + 1$ on the elements of the mesh used in the reconstruction.

Using these methods to compute the error of the DGVEM1 solutions we get Figure 6.5, while for DGVEM2 solutions we get Figure 6.6.

Looking at the error in both figures, we see that the polynomial projection slightly underestimates the error compared to the reconstruction, but this effect is relatively small. Especially when we compare it to the VEM interpolation, which both overestimates and underestimates the error depending on the tile. This effect is quite severe for the DGVEM2 on boat tiles (overestimate by almost a factor 4) or DGVEM1 and triangles (underestimate by a factor 6).

Moreover, looking at the slope of the lines, we can see that both the polynomial projection and solution reconstruction are relatively close to the expected convergence rate. While the error estimate based on the VEM interpolation again less predictable, with slightly varying slope.

When looking at the runtime, we see as expected that the solution reconstruction approach is extremely expensive, costing about 10 (DGVEM1) or 100 (DGVEM2) times as much time as the other options. The actual cost is probably even higher, as we run the reconstruction on 4 cores in parallel. Comparing the VEM interpolation with the polynomial projection, we see that the former is less expensive.

Comparing the two practical approaches, there is no obvious best method. We expect that the polynomial projection is better in practice, as it shows a more consistent and accurate behaviour. Moreover, the accuracy of this method could be improved by projecting the exact solution u to polynomials of order larger than k, or even not projecting it at all and using triangulation and quadrature.

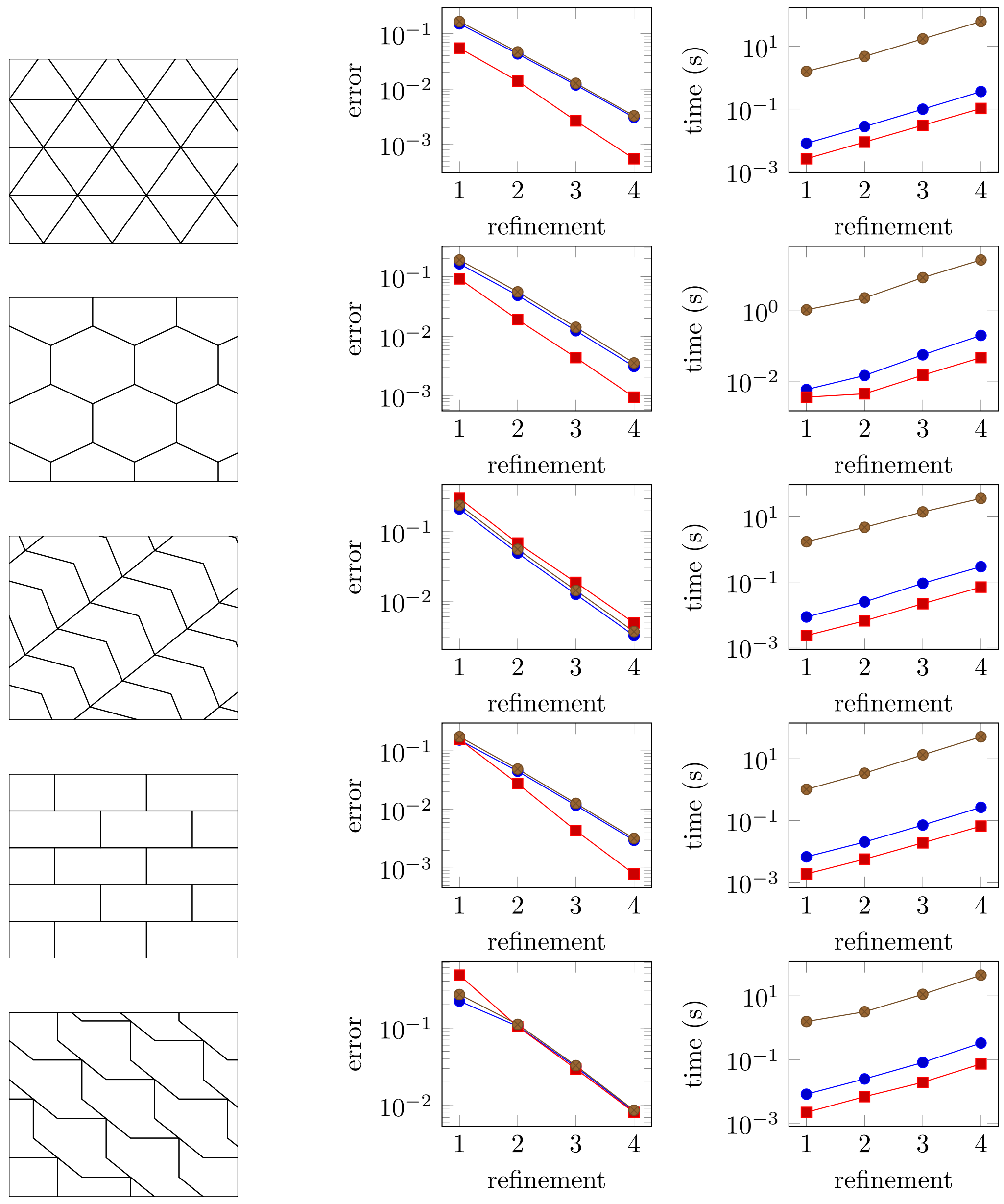


Figure 6.5: Error computations using different approaches for DGVEM1, similar to Figure 6.3 for ADG2. In blue the polynomial projection error, in red the VEM interpolant error and in brown the error computed by reconstructing the solution.

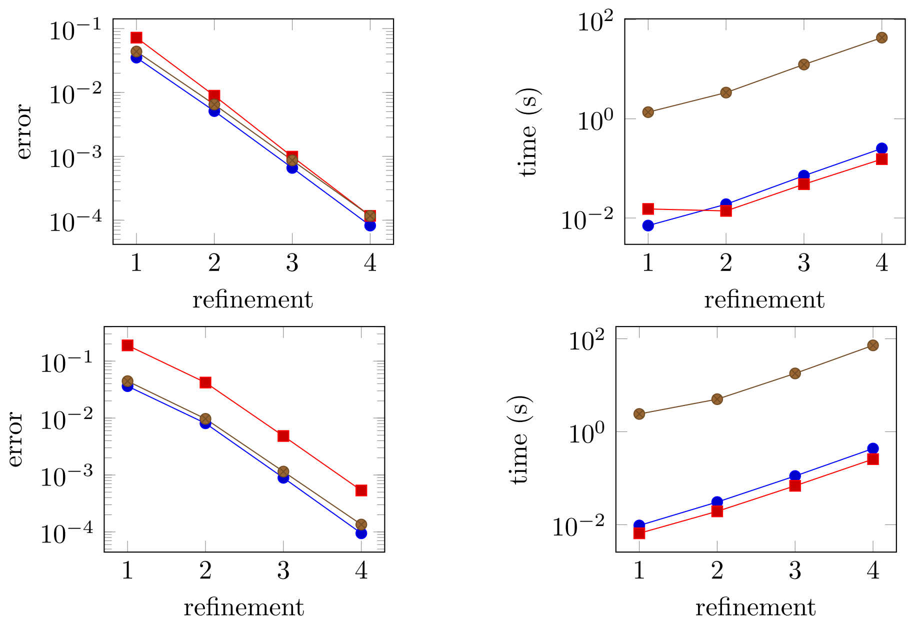


Figure 6.6: Error computation using different approaches for DGVEM2. Results are similar to Figure 6.5. On the top two figures are for the hexagon mesh, while the bottom two figures use the boat tile mesh.

6.3.2 RHS

In addition to the norm computation, we also discussed several alternatives for the element vector computation in Chapter 5. We consider three alternatives for the numerical tests:

- Polynomial projection of f, using as usual (4.18) for the computation of the moments. Then using (5.10) for the actual computation of the element vector.
- Using the VEM interpolation of f, and computing the element vector using (5.11). As before we use (4.18) for the require moments.
- Using a local Taylor polynomial of order k for f, then using (5.10) for the computation of the element vector.

As usual, we applied these methods to the standard sine test case. We computed the error in the resulting solution using the reconstruction approach. Additionally, we measured the time used for computing the element vector. The result of these computations is shown in Figure 6.7.

We observe that the Taylor polynomial and polynomial projection approaches both give consistent (small) errors in the result, with the Taylor polynomial approach giving the smallest. Just as with the norm computation, we see that the approach using the VEM interpolant behaves more eratic. Moreover, it usually results in larger errors in the solution than the other two methods. We expect that the quality of the VEM interpolant is quite dependant on the mesh.

Looking at the computational time of the various methods, we see that the Taylor polynomial

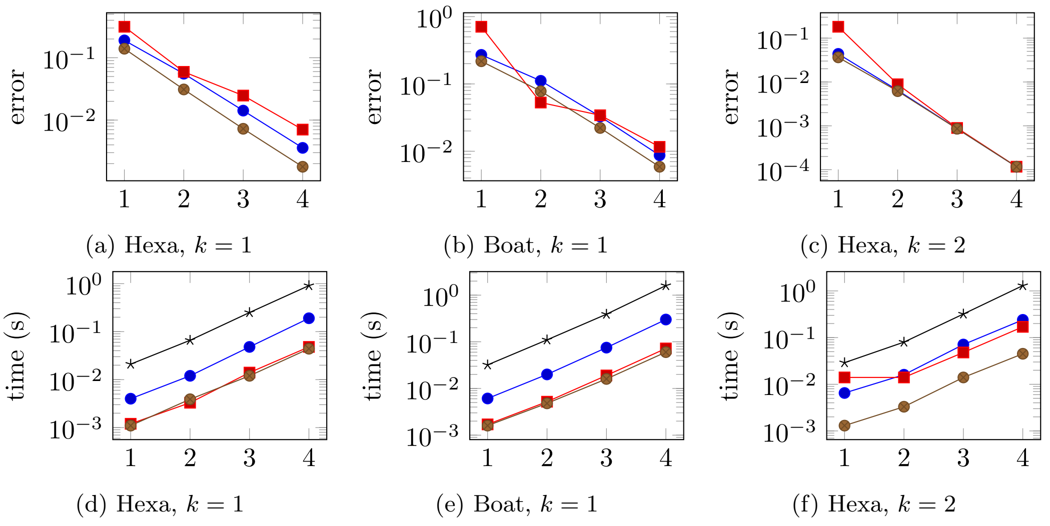


Figure 6.7: Comparison of the element vector computation for DGVEM. In blue the polynomial projection, in red the VEM interpolation and in brown the Taylor polynomial. The black line corresponds to the time for the computation using DGVEM, but with a zero element vector.

approach is, as expected, the fastest. For DGVEM1, the VEM interpolant is about the same speed as the Taylor polynomial, this is not surprising as for $k = 1$ the VEM space only has vertex degrees of freedom. Hence, it only needs to compute some values of f and, more expensive, the mass matrix M. Contrastingly, for DGVEM2, we need a single moments of f for the interpolant, the time of the VEM interpolant approach is in this case also much higher than that of the Taylor polynomial approach.

Comparing these runtimes with the runtime of DGVEM with a zero element vector, we see that the computation of the element vector can be a significant (additional) cost. Observe that the polynomial projection takes about 15 to 20 percent, and Taylor polynomial only 3 to 5 percent. Therefore, we conclude that using a cheaper method, like the Taylor polynomial approach, can be a good optimization.

Combining all these arguments, we see that the Taylor polynomial approach is both the most efficient and usually the most accurate for our test case. Therefore it seems to be the most interesting to use. Contrastingly, the VEM interpolant is not so interesting, as it results in erratic and usually large errors, especially when compared to the other methods.

6.4 Comparing DGVEM and ADG

With all the testing of the various element vector and error norm computations done, we now turn to comparing the results for ADG and DGVEM. We start with explaining the difficulties with comparing them, followed by a basic description of the test setup. Using the actual tests, we will compare ADG2 and DGVEM1 for several problems. Finally, we will summarize these results.

6.4.1 Difficulties

There are two difficulties in comparing ADG with DGVEM. Firstly, because the basis functions of DGVEM are virtual, we do not have a good way to plot the solution. Secondly, a fair comparison of the methods is harder due to the different number of degrees of freedom, er element for the different methods.

To solve the first problem of plotting the DGVEM solution, we can take two routes. We could reconstruct the solution and draw this result. As this is very expensive, and thus not practical for prototyping, we use a simpler approximation. As coloring of the solution requires that we only draw triangles. We use the following steps to generate the drawing triangles for both ADG and DGVEM to compute the triangles in the plot:

- We add to each edge $k-1$ equidistant points, to approximate the non-linear solution on each edge.
- We construct triangles by drawing lines from the centroid of the element to both the vertices and the $k-1$ equidistant points on edge. This results a triangle fan around the centroid.
- For both methods we can compute the values at the boundary points directly. For ADG, we can also compute the value at the centroid, while for DGVEM we approximate it by using the constant part of $\Pi^{\nabla} u_h$.[1]

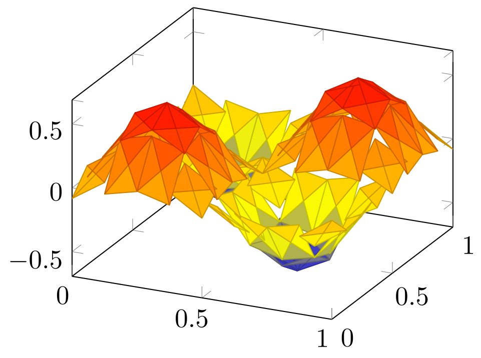
To see the effect of the second problem, the difference in the number of degrees of freedom, we look at Figures 6.8a and 6.8b. On each square element ADG1 has only three degrees of freedom, thereby forcing the gaps in the solution. With the four conforming basis functions of DGVEM1 a solution without gaps is possible. As result of the smoother solution we see a much closer approximation of the theoretical solution.

We can prevent this mismatch in the degrees of freedom, by carefully selecting the order of both DGVEM, ADG and the number of vertices in each element. Using ADG1 we are limited by its three basis functions to triangles and DGVEM1, for which traditional SIPG is a much better choice. Considering ADG2 (six basis functions), we can match it to DGVEM1 on an element with six vertices. A match for ADG3 (ten basis functions) is limited to DGVEM1 with an element with ten vertices. While ADG4 (fifteen basis functions) can be used with DGVEM1, DGVEM2 or DGVEM3 using an element with 15, 7 and 4 vertices, respectively.

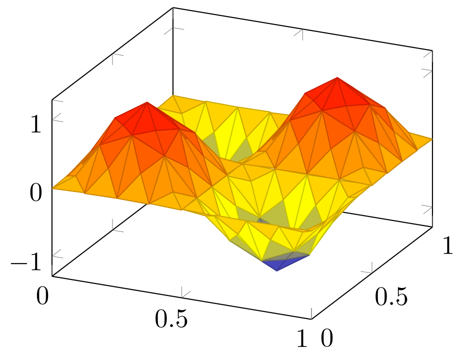
As an example, we compare DGVEM1 with ADG2 on elements with six vertices, Figures 6.8c and 6.8d. Note that unlike Figure 6.8b the solution of both methods is close to the theoretical maximum, having similar L^2-errors of 0.01 (ADG2) and 0.03 (DGVEM1).

Comparing the convergence rates of ADG2 and DGVEM1 from the previous section in Figure 6.9, we clearly see the effect of the higher convergence rate of ADG2. Thus, in a traditional setting with small enough mesh size, ADG2 will be more accurate than DGVEM1. However, for our application the problem is that the mesh size is already too small for normal elements. Thus, we can not make the assumption that higher convergence rate of ADG2 results in a more accurate solution than DGVEM1.

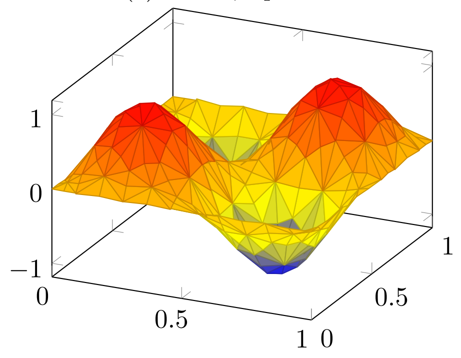
[1]It would be more logical to use $\Pi^0_k u_h$ instead of $\Pi^{\nabla} u_h$. This code was written before we implemented Π^0_k, and we, unfortunately, discovered this oversight too late to change it.



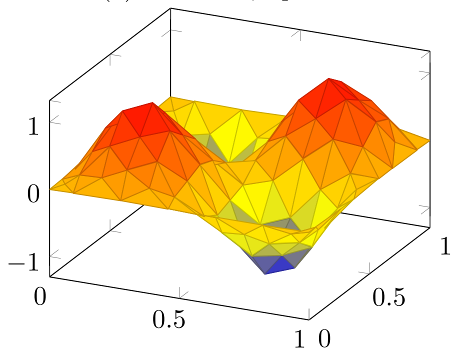
(a) ADG1, square mesh



(b) DGVEM1, square mesh



(c) ADG2, hexa mesh



(d) DGVEM1, hexa mesh

Figure 6.8: Comparison of the solutions of ADG and DGVEM for the sine test case. On top we have ADG1 and DGVEM1 on a square mesh. Note that the ADG1 solution not only has large gaps, but also has a maximum amplitude of about 0.57 instead of the theoretical 1. On the bottom the third order ADG2 solution and the DGVEM1 solution, but now on a hexagon mesh are shown. Both have similar errors of 0.01 and 0.03 respectively. For the computation of the element vector we used triangulation and quadrature for ADG and the Taylor polynomial for DGVEM.

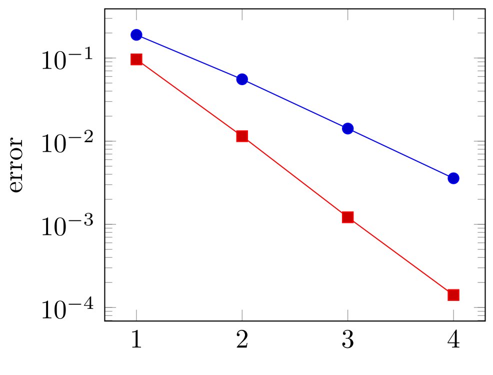


Figure 6.9: Comparison of the convergence of DGVEM1 with that of ADG2 on a hexagon mesh. In blue the error computed using the solution reconstruction approach for DGVEM1, in red the triangulation and quadrature error of ADG2.

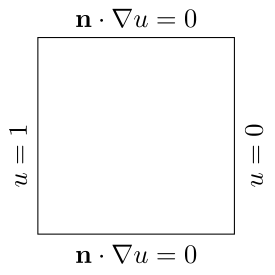

$\mathbf{n} \cdot \nabla u = 0$
$u = 1$
$u = 0$
$\mathbf{n} \cdot \nabla u = 0$

Figure 6.10: Boundary conditions of the test cases to compare ADG with DGVEM. In the figure $\mathbf{n}$ is the outward pointing normal.

6.4.2 Test setup

To compare ADG2 and DGVEM1 for our problem, we need a more realistic scenario than the sin test. As our original problem is the result of internal interfaces with small geometric details, we try to recreate such a situation as test case here.

Specifically, we use the unit square as domain with both Dirichlet and Neumann boundary conditions. For $x_1 = 0$ we set $u = 1$, and for $x_1 = 1$ we set $u = 0$. On the other boundaries ($x_2 = 0, 1$) we apply zero Neumann conditions, see Figure 6.10 for a sketch. Furthermore, we set the source term f to zero, preventing all the trouble with computing it. On the domain, we create interface by setting κ, on each element, to either $\kappa = 10^{-2}$ or $\kappa = 1$.

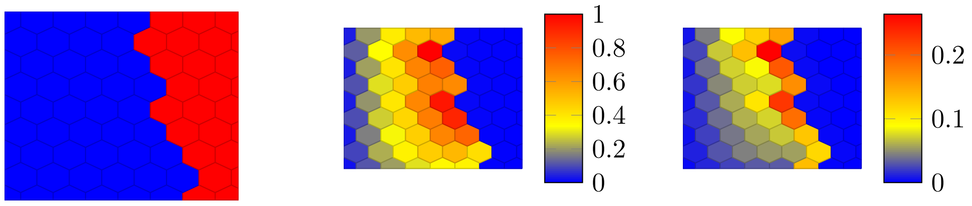
As a consequence of this varying κ, we do not have an expression for the actual solution. Therefore, we will compute a reference solution using SIPG3 on mesh constructed by triangulating each element on the original mesh. The error in this reference solutions is, due to the finer mesh and the higher order, expected to be far smaller than the error in either the ADG2 or DGVEM1 solutions on the original mesh. To subsequently compute the error in the ADG2 solution, we use a quadrature rule on the triangular mesh of the reference solution, while for the DGVEM1 solution we reconstruct the solution and use quadrature on the mesh of the reconstruction.

6.4.3 Tests

For the actual tests, we look at three different interface problems.

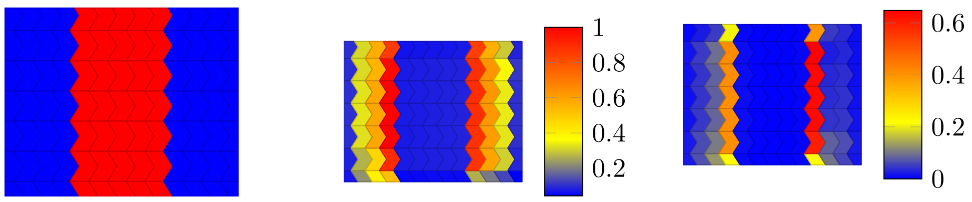
In the first test, we look at the general effects of an interface on the error of ADG1 and DGVEM1. We constructed, as Figure 6.11a shows, a mesh of hexagons, with two regions with different κ. The local error $\|u - u_h\|_E$ is shown in Figures 6.11b and 6.11c. We can clearly see that the elements of ADG2 have a far larger error near the interface. This is also represented in the error in the solution over the whole domain ($\|u - u_h\|_\Omega$), 7.1×10^{-3} for ADG2 and 1.4×10^{-3} for DGVEM1.

As second test, we change from the hexagonal tiles to the non-convex arrow ones. The results in Figure 6.12 show again a much larger error for ADG2. Though in this case the difference in the total error is much smaller, 5.1×10^{-3} for ADG2 compared to 2.3×10^{-3} for DGVEM1. Moreover, when moving away from the interface we see that the local error diminishes much slower for ADG2 than for DGVEM1.



(a) Domain (b) ADG2 (c) DGVEM1

Figure 6.11: Comparison of ADG2 and DGVEM1 for a domain with an interface. In blue the elements with $\kappa = 10^{-2}$, while in red those with $\kappa = 1$. The figures for ADG2 and DGVEM1 show the L^2-error for each element. The value on the colorbars are relative to the maximum error of a single element in ADG2 solution, 0.0022. Note that the colorbar for the DGVEM1 solution stops at bout 0.25.



(a) Domain (b) ADG2 (c) DGVEM1

Figure 6.12: Comparison of ADG2 and DGVEM1 for a domain with a vein with small κ. Again in blue the elements with $\kappa = 10^{-2}$, while in red those with $\kappa = 1$. The figures for ADG2 and DGVEM1 show the L^2-error on each element. The values on the colorbar are relative to the maximum error of on a single element in the ADG2 solution, 0.0013. Note that the colorbar of the DGVEM solution stops at about 0.62.

As third test case, we look at the possible overshoot and undershoot of the solution. Specifically, as we solve a version of the Laplace equation, we expect that the solution does not have any local maxima or minima inside the domain and that the global extrema are on the boundary. A finite element solution does not necessarily satisfy these properties and could even have a more extreme global maximum or minimum than the boundary condition. Both are generally unwanted, as not satisfying these criteria could lead to violating physical constraints (e.g. a concentration should be positive, but the method giving a negative one).

To see the difference in this effect clearly, we create very sharp gradient in our solution, see Figure 6.13. We clearly observe a more jagged edge in ADG2 solution when compared to DGVEM1. Using the values of the plot, we see that the solution of ADG2 overshoots the theoretical maximum of 1 by 0.013, while undershooting the minimum (0) by -0.026. Much larger than the overshoot of 0.005 and undershoot of -0.0015 in the DGVEM1 solution.

6.5 Summary

In this chapter we looked from a numerical point of view to the ADG and DGVEM methods. After standard convergence tests to verify the implementation, we considered alternatives for the element vector and error norm computation to determine the best candidates. Furthermore, we compared ADG and DGVEM to determine which numerical discretisation perform best.

We considered several numerical tests for both ADG and DGVEM, both to determine the performance of the alternative approaches for the element vector and the error norm computation. Moreover, we compared ADG for $k = 2$ to DGVEM for $k = 1$.

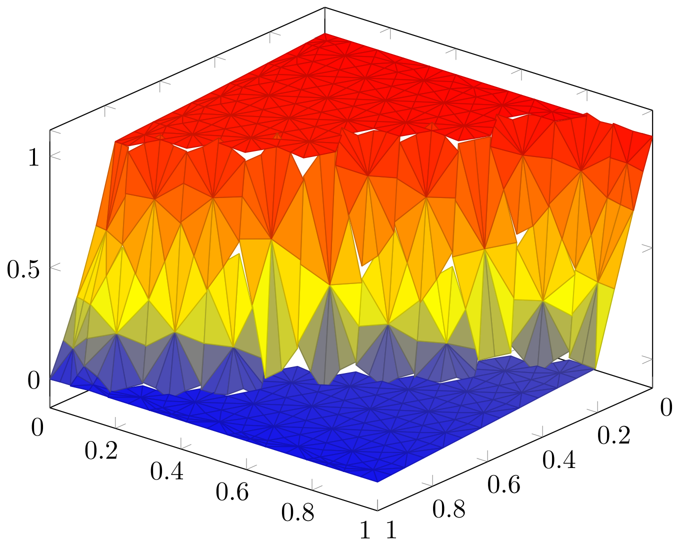
For ADG, we saw that none of the alternatives for the error computations were much better than using triangulation and quadrature. The Taylor polynomial was either much less accurate or much more expensive, while the polynomial projection was more expensive and only slightly less accurate.

For the element vector computation in ADG ($k = 2$), we saw that the use of a Taylor polynomial did affect the computational time. On triangular meshes its performance was similar to using quadrature, while on a mesh created using the boat tile elements, the Taylor polynomial approach reduced the computational cost of the element vector computation to about 30% of the quadrature approach.

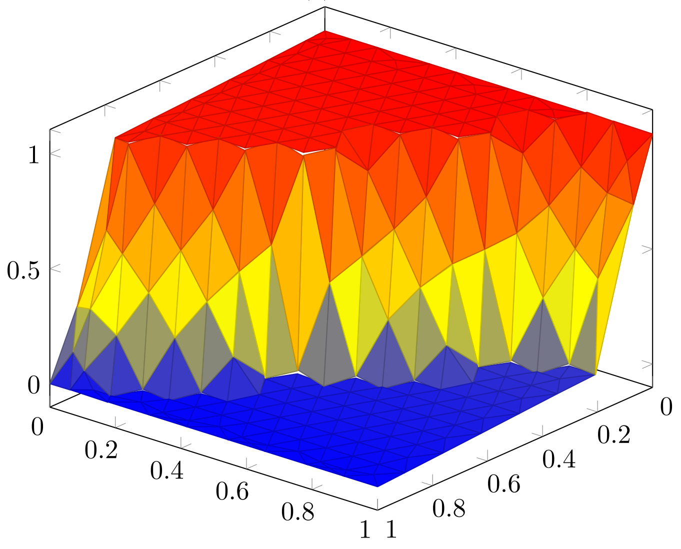
We saw a similar effect with DGVEM, where the Taylor polynomial approach was fastest, and surprisingly, also the most accurate. The VEM interpolation was comparatively a far less useful approach, it increases the error and behaves more erratic, thereby complicating convergence studies. A similar result was obtained when using this method for the error computation.

To comparing DGVEM and ADG, we selected ADG2 and DGVEM1 on tiles with six vertices. This ensures that both have the same number of degrees of freedom per element. As a consequence ADG2 has, as second order method, a much better convergence rate than DGVEM1, a first order method. However, this requires a small enough mesh, which is not so useful as our main problem is, that the mesh is already too small.

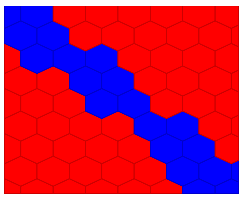
For a comparison more practical to our problem, we looked at three artificial interface problems. For these test cases, we saw that DGVEM1 can have much lower errors compared to ADG1. Moreover, that the VEM basis functions can also be used in a conforming manner results in the difference in Figure 6.13. In this figure we compared the overshoot and undershoot of ADG2



(a) ADG2



(b) DGVEM1



(c) Domain

Figure 6.13: Comparison of the ADG2 and DGVEM1 computation for a very steep solution. The blue elements have $\kappa = 10^{-2}$, and those in red $\kappa = 1$.

and DGVEM1, with the DGVEM1 solution showing far smaller gaps and its maximum and minimum being closer to their theoretical values.

Chapter 7

Conclusions

In this thesis we investigated two finite element methods that allow polygonal elements in combination with a Discontinuous Galerkin approach. The idea is that such elements could be used to reduce the impact of a geometrically complex interface in the domain on the computational cost.

Extending the standard finite element approach to polygonal elements does, as already noted in the introduction, result in some problems. The most important one is that one has to choose between using polynomial or conforming basis functions, as polynomial conforming basis functions do not exist on general polygons or polyhedra. The two methods that we selected from literature for further study, make a fundamentally different choice for this problem. ADG uses polynomial basis functions, while DGVEM uses the conforming basis functions of VEM.

We described both methods in detail for a generalized Poisson problem. In this description we paid particular attention to another problem of polygonal elements: the lack of an efficient element quadrature rule. We used the ideas from VEM to integrate polynomials on the element by converting them into boundary integrals, which is sufficient for the element matrix construction of both methods. The element vector and error computations, are more difficult, as the source term and solution and in general non-polynomial. We developed several alternative approaches, with varying accuracy and computational cost, to compute the element vector and error.

From the numerical tests using these alternatives, we can draw several conclusions. For ADG, we saw that triangulating the polygonal element and using quadrature on each triangle is not as computationally expensive as expected. Combine this with the simplicity and flexibility of this approach and it is clear why this is the preferred method. Of the alternatives, only the Taylor polynomial approach for the element vector computation proved to be interesting. It can, for polygonal elements, significantly reduce the computational cost, without significant changes to the error. But as downside we have that the accuracy is harder to predict and not all source terms allow the construction of the Taylor polynomial.

The numerical tests using DGVEM have a less clear conclusion. For the error computation, we saw that the polynomial projection of the error gives a more accurate value and more consistent convergence rates than the VEM interpolation, while being far less expensive than the reconstruction of the solution to perform standard quadrature. For the element vector computation, we again obtained good accuracy and speed when using a local Taylor polynomial.

With the polynomial projection proving again to be a more stable and expensive approach than the VEM interpolation.

Comparing DGVEM with ADG is more complicated. In general we saw that DGVEM uses more degrees of freedom than ADG for the same polynomial accuracy. On a sufficiently refined mesh ADG could, compared to DGVEM, therefore obtain a better accuracy with the same number of degrees of freedom. However, for our application the problem is that we try to get a mesh with larger, not smaller, elements. Therefore, we compared ADG and DGVEM for several interface problems, where we carefully choose the parameters so that ADG and DGVEM use the same number of degrees of freedom on the average element.

The results from these test cases were quite clear. Even with the extra order of accuracy of ADG, it had larger errors in the solution than DGVEM. Moreover, the overshoot above the theoretical maximum and undershoot below the theoretical minimum were an order higher for ADG compared to DGVEM.

For our intended application we consider the following. DGVEM has better accuracy near the interface and allows more flexible finite element spaces. ADG does have a higher convergence order per degree of freedom, but this is of not much use for the larger elements in our application. Therefore, we recommend the use of DGVEM for our application.

7.1 Discussion and future work

Having answered the main research question it is time to reflect on the work done and to look forward to possible use and future research.

We start by, that even though we recommend DGVEM, we also have to recommend ADG. While not the best method to use in the final application, it is a very useful method as intermediate step to building DGVEM. We implemented ADG first, which required the implementation of all the machinery for polygonal meshes, scaled monomials, etc. while it still is easy to evaluate basis functions. Then, as second step, we implemented the more complicated projection operators and matrices required in DGVEM. This process reduced the amount of code where a bug could possibly be.

Secondly, we are, as far as we can tell the first to compare DGVEM with ADG, and possibly also the first to implement DGVEM. Comparing them in a fair manner is quite hard, especially when we want a fair measure of the computational cost. We chose here to compare them on elements with the same number of degrees of freedom, as this is independent of how optimal our implementation is. In reality we would be more limited by computational time and memory use. However, these are far more difficult to measure and optimize in a MATLAB prototype and are also dependent on the PDE that is solved.

Thirdly, having seen all the complexity needed in DGVEM, we do not expect it to be quite a bit slower than traditional SIPG. Additionally, the element vector and error computation are relatively fragile and sometimes show higher or more erratic convergence rates. Therefore we recommend using a mesh with mostly triangular elements, while using the polygonal elements only where they are most useful. That way one combine the fast SIPG with the more complex and expensive DGVEM only on those elements that need it. For example using them on the detailed interface or on elements with hanging nodes.

As fourth point, while we looked at the computation of the efficient error norm and element vector, this was out of necessity, since we did not find much literature on how these procedures are implemented. There are probably other methods that strike a different balance between accuracy and computational expense. Moreover, our experiment with the solution reconstruction of DGVEM provided valuable results, but needs more theoretical background, especially considering regularity at reflex corners of non-convex elements.

Finally, we recommend against using large agglomerated meshes as used in some of the ADG literature. These agglomerated meshes usually have rough element boundaries with many edges, resulting in computationally expensive element integrals. Which is expensive independent of whether one uses the non-agglomerated elements with quadrature, triangulation and quadrature or constructs the inner product matrix. For cheap element quadrature, it is best to reduce the number of edges of the elements as much as possible.

Bibliography

[1] METIS - serial graph partitioning and fill-reducing matrix ordering. `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`. Accessed on 2017-08-11.

[2] Mgridgen. `http://www-users.cs.umn.edu/~moulitsa/software.html`. Accessed on 2017-08-11.

[3] Bashir Ahmad, Ahmed Alsaedi, Franco Brezzi, L Donatella Marini, and A Russo. Equivalent projectors for virtual element methods. *Computers & Mathematics with Applications*, 66(3):376–391, 2013.

[4] Odd Andersen, Halvor M Nilsen, and Xavier Raynaud. Virtual element method for geomechanical simulations of reservoir models. *Computational Geosciences*, 21(5-6):866–893, 2017.

[5] Paola F Antonietti, Andrea Cangiani, Joe Collis, Zhaonan Dong, Emmanuil H Georgoulis, Stefano Giani, and Paul Houston. Review of discontinuous Galerkin finite element methods for partial differential equations on complicated domains. In *Building Bridges: Connections and Challenges in Modern Approaches to Numerical Partial Differential Equations*, pages 279–308. Springer, 2016.

[6] Paola F Antonietti, L Beirao Da Veiga, Simone Scacchi, and Marco Verani. A C^1 virtual element method for the Cahn–Hilliard equation with polygonal meshes. *SIAM Journal on Numerical Analysis*, 54(1):34–56, 2016.

[7] Paola F Antonietti, Stefano Giani, and Paul Houston. hp-version composite discontinuous Galerkin methods for elliptic problems on complicated domains. *SIAM Journal on Scientific Computing*, 35(3):A1417–A1439, 2013.

[8] Douglas N Arnold, Franco Brezzi, Bernardo Cockburn, and L Donatella Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM journal on numerical analysis*, 39(5):1749–1779, 2002.

[9] Francesco Bassi, L Botti, Alessandro Colombo, and Stefano Rebay. Agglomeration based discontinuous Galerkin discretization of the Euler and Navier–Stokes equations. *Computers & Fluids*, 61:77–85, 2012.

[10] Francesco Bassi, Lorenzo Botti, Alessandro Colombo, Daniele A Di Pietro, and Pietro Tesini. On the flexibility of agglomeration based physical space discontinuous Galerkin discretizations. *Journal of Computational Physics*, 231(1):45–65, 2012.

[11] Éric Béchet, Nicolas Moës, and Barbara Wohlmuth. A stable Lagrange multiplier space for stiff interface conditions within the extended finite element method. *International Journal for Numerical Methods in Engineering*, 78(8):931–954, 2009.

[12] L Beirão da Veiga, F Brezzi, A Cangiani, G Manzini, LD Marini, and A Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214, 2013.

[13] L Beirão da Veiga, F Brezzi, LD Marini, and A Russo. The hitchhiker's guide to the virtual element method. *Mathematical models and methods in applied sciences*, 24(08):1541–1573, 2014.

[14] Stefano Berrone and Andrea Borio. Orthogonal polynomials in badly shaped polygonal elements for the virtual element method. *Finite Elements in Analysis and Design*, 129:14–31, 2017.

[15] F Brezzi, Richard S Falk, and L Donatella Marini. Basic principles of mixed virtual element methods. *ESAIM: Mathematical Modelling and Numerical Analysis*, 48(4):1227–1240, 2014.

[16] F Brezzi and LD Marini. Virtual element and discontinuous Galerkin methods. In *Recent developments in discontinuous Galerkin finite element methods for partial differential equations*, pages 209–221. Springer, 2014.

[17] Franco Brezzi and L Donatella Marini. Virtual element methods for plate bending problems. *Computer Methods in Applied Mechanics and Engineering*, 253:455–462, 2013.

[18] Erik Burman and Peter Hansbo. Fictitious domain finite element methods using cut elements: I. A stabilized Lagrange multiplier method. *Computer Methods in Applied Mechanics and Engineering*, 199(41):2680–2686, 2010.

[19] Andrea Cangiani, Zhaonan Dong, Emmanuil H Georgoulis, and Paul Houston. hp-version discontinuous Galerkin methods for advection-diffusion-reaction problems on polytopic meshes. *ESAIM: Mathematical Modelling and Numerical Analysis*, 50(3):699–725, 2016.

[20] Andrea Cangiani, Emmanuil H Georgoulis, and Paul Houston. hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes. *Mathematical Models and Methods in Applied Sciences*, 24(10):2009–2041, 2014.

[21] Philippe G Ciarlet. *The Finite Element Method for Elliptic Problems*, volume 40. SIAM, 2002.

[22] Joe Collis and Paul Houston. Adaptive discontinuous Galerkin methods on polytopic meshes. In *Advances in Discretization Methods*, pages 187–206. Springer, 2016.

[23] Alessandro Colombo. *An agglomeration-based discontinuous Galerkin method for compressible flows*. PhD thesis, Università degli studi di Bergamo, 2011.

[24] L Beirão Da Veiga, Franco Brezzi, L Donatella Marini, and A Russo. $H(\mathrm{div})$ and $H(\mathbf{curl})$-conforming virtual element methods. *Numerische Mathematik*, 133(2):303–332, 2016.

[25] Lourenco Beirão da Veiga and Gianmarco Manzini. A virtual element method with arbitrary regularity. *IMA Journal of Numerical Analysis*, 34(2):759–781, 2013.

[26] Gautam Dasgupta. Interpolants within convex polygons: Wachspress' shape functions. *Journal of Aerospace Engineering*, 16(1):1–8, 2003.

[27] D. Devashish, Shakeeb B. Hasan, J. J. W. van der Vegt, and Willem L. Vos. Reflectivity calculated for a three-dimensional silicon photonic band gap crystal with finite support. *Phys. Rev. B*, 95:155141, Apr 2017.

[28] Jim Douglas and Todd Dupont. Interior penalty procedures for elliptic and parabolic galerkin methods. In *Computing methods in applied sciences*, pages 207–216. Springer, 1976.

[29] Alexandre Ern, Annette F Stephansen, and Paolo Zunino. A discontinuous galerkin method with weighted averages for advection–diffusion equations with locally small and anisotropic diffusivity. *IMA Journal of Numerical Analysis*, 29(2):235–256, 2008.

[30] Michael S Floater. Mean value coordinates. *Computer aided geometric design*, 20(1):19–27, 2003.

[31] Michael S Floater. Wachspress and mean value coordinates. In *Approximation Theory XIV: San Antonio 2013*, pages 81–102. Springer, 2014.

[32] Michael S Floater. Generalized barycentric coordinates and applications. *Acta Numerica*, 24:161–214, 2015.

[33] Michael S Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.

[34] Thomas-Peter Fries and Ted Belytschko. The extended/generalized finite element method: an overview of the method and its applications. *International Journal for Numerical Methods in Engineering*, 84(3):253–304, 2010.

[35] Stefano Giani and Paul Houston. hp–adaptive composite discontinuous Galerkin methods for elliptic problems on complicated domains. *Numerical Methods for Partial Differential Equations*, 30(4):1342–1367, 2014.

[36] Diana Grishina. *3D Silicon Nanophotonics*. PhD thesis, University of Twente, 7 2017.

[37] Sjoerd A Hack, Jaap JW van der Vegt, and Willem L Vos. Three-dimensional crystal of cavities in a 3d photonic band gap crystal. In *Lasers and Electro-Optics Europe & European Quantum Electronics Conference (CLEO/Europe-EQEC, 2017 Conference on)*, pages 1–1. IEEE, 2017.

[38] Wolfgang Hackbusch and Stefan A Sauter. Composite finite elements for problems containing small geometric details. *Computing and Visualization in Science*, 1(1):15–25, 1997.

[39] Wolfgang Hackbusch and Stefan A Sauter. Composite finite elements for the approximation of PDEs on domains with complicated micro-structures. *Numerische Mathematik*, 75(4):447–472, 1997.

[40] Kai Hormann and Natarajan Sukumar. Maximum entropy coordinates for arbitrary polytopes. In *Computer Graphics Forum*, volume 27, pages 1513–1520. Wiley Online Library, 2008.

[41] P Milbradt and T Pick. Polytope finite elements. *International Journal for Numerical Methods in Engineering*, 73(12):1811–1835, 2008.

[42] Irene Moulitsas and George Karypis. Multilevel algorithms for generating coarse grids for multigrid methods. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 45–45. ACM, 2001.

[43] SE Mousavi, H Xiao, and N Sukumar. Generalized Gaussian quadrature rules on arbitrary polygons. *International Journal for Numerical Methods in Engineering*, 82(1):99–113, 2010.

[44] Sundararajan Natarajan, Stéphane Bordas, and D Roy Mahapatra. Numerical integration over arbitrary polygonal domains based on Schwarz–Christoffel conformal mapping. *International Journal for Numerical Methods in Engineering*, 80(1):103–134, 2009.

[45] Vinh Phu Nguyen, Timon Rabczuk, Stéphane Bordas, and Marc Duflot. Meshless methods: a review and computer implementation aspects. *Mathematics and computers in simulation*, 79(3):763–813, 2008.

[46] Gilbert Strang and George J Fix. *An analysis of the finite element method*, volume 212. Prentice-hall Englewood Cliffs, NJ, 1973.

[47] N Sukumar. Construction of polygonal interpolants: a maximum entropy approach. *International Journal for Numerical Methods in Engineering*, 61(12):2159–2181, 2004.

[48] N Sukumar and EA Malsch. Recent advances in the construction of polygonal finite element interpolants. *Archives of Computational Methods in Engineering*, 13(1):129–163, 2006.

[49] N Sukumar and A Tabarraei. Conforming polygonal finite elements. *International Journal for Numerical Methods in Engineering*, 61(12):2045–2066, 2004.

[50] A Tabarraei and N Sukumar. Extended finite element method on polygonal and quadtree meshes. *Computer Methods in Applied Mechanics and Engineering*, 197(5):425–438, 2008.

[51] Pietro Francesco Tesini. *An h-multigrid approach for high-order discountinuous Galerkin methods*. PhD thesis, Università degli studi di Bergamo, 2008.

[52] Giuseppe Vacca and Lourenco Beirão da Veiga. Virtual element methods for parabolic problems on polygonal meshes. *Numerical Methods for Partial Differential Equations*, 31(6):2110–2134, 2015.

[53] Eugene L Wachspress. A rational basis for function approximation. *Lecture notes in mathematics*, 228:223–252, 1971.

[54] Hong Xiao and Zydrunas Gimbutas. A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions. *Computers & mathematics with applications*, 59(2):663–676, 2010.