Overcoming Element Quality Dependence of Finite Elements with Adaptive Extended Stencil FEM (AES-FEM)

Rebecca Conley, Tristan J. Delaney and Xiangmin Jiao

Department of Applied Mathematics & Statistics, Stony Brook University, Stony Brook, NY 11794

SUMMARY

The finite element methods (FEM) are important techniques in engineering for solving partial differential equations, but they depend heavily on element shape quality for stability and good performance. In this paper, we introduce the *Adaptive Extended Stencil Finite Element Method* (AES-FEM) as a means for overcoming this dependence on element shape quality. Our method replaces the traditional basis functions with a set of *generalized Lagrange polynomial (GLP) basis functions*, which we construct using local weighted least-squares approximations. The method preserves the theoretical framework of FEM, and allows imposing essential boundary conditions and integrating the stiffness matrix in the same way as the classical FEM. In addition, AES-FEM can use higher-degree polynomial basis functions than the classical FEM, while virtually preserving the sparsity pattern of the stiffness matrix. We describe the formulation and implementation of AES-FEM, and analyze its consistency and stability. We present numerical experiments in both 2D and 3D for the Poison equation and a time-independent convection-diffusion equation. The numerical results demonstrate that AES-FEM is more accurate than linear FEM, is also more efficient than linear FEM in terms of error versus runtime, and enables much better stability and faster convergence of iterative solvers than linear FEM over poor-quality meshes. Copyright © 2016 John Wiley & Sons, Ltd.

1. INTRODUCTION

The finite element methods (FEM) are arguably one of the most important numerical tools for solving partial differential equations (PDE) over complex domains in engineering. They account for an overwhelming majority of the commercial and research code for modeling and simulations, and there is a vast amount of theoretical work to provide a rigorous foundation; see e.g., [1].

Despite their apparent success in many applications, classical finite element methods have a very fundamental limitation: *they are dependent on element shape quality.* This is especially true for elliptic and parabolic problems, for which the resulting linear system is often ill-conditioned if a

---

*Correspondence to: jiao@ams.sunysb.edu

mesh contains a few "bad" elements. This can lead to very slow convergence of iterative solvers and sometimes even a loss of accuracy. Because of this, researchers and users of FEM often spend a tremendous amount of time and computing power to generate and maintain meshes, trying to fix that one last bad element. This has spurred much successful research in meshing, such as Delaunay triangulation [2, 3], advancing front [4], octree-based methods [5], etc. However, the meshing problem continues to become more and more challenging as applications become more and more sophisticated and demanding, and also with the increased use of higher-order methods, such as spectral element methods [6], discontinuous Galerkin methods [7, 8], etc.

The FEM community has long considered this dependency on element quality as a critical issue, and the community has been actively searching for alternative methods to mitigate the issue for decades. Examples of such alternative methods include the diffuse element or element-free Galerkin methods [9, 10], least-squares FEM [11], generalized or meshless finite different methods [12, 13, 14, 15, 16], generalized or extended FEM [17, 18], and partition-of-unity FEM [19]. To reduce the dependency on mesh quality, these methods avoid the use of the piecewise-polynomial Lagrange basis functions found in the classical FEM. However, they also lose some advantages of the classical FEM. In particular, for the generalized finite different methods, the strong form instead of the weak form of the PDE must be used. The partition-of-unity FEM and other similar generalizations often incur complexities in terms of imposing essential boundary conditions and/or integrating the stiffness matrix [19]. Therefore, it remains an open problem to develop a numerical method that overcomes the element-quality dependence, while preserving the theoretical framework of FEM, without complicating the imposition of boundary conditions and numerical integration.

This paper introduces a new method, called the *Adaptive Extended Stencil Finite Element Method* (*AES-FEM*) (pronounced as ace-F-E-M), to address this open problem. Similar to some of the aforementioned methods, the AES-FEM replaces the piecewise-polynomial Lagrange basis functions in the classical FEM with alternative basis functions. Different from those methods, our basis functions are partition-of-unity polynomial basis functions, constructed based on local weighted least squares approximations, over an adaptively selected stencil to ensure stability. We refer to these basis functions as *generalized Lagrange polynomial* (*GLP*) basis functions. Another difference of AES-FEM from most other generalizations of FEM is that AES-FEM preserves the traditional finite element shape functions as the weight functions (a.k.a. test functions) in the weak form, to preserve the compact support of integration and the weak form after integration by parts. This combination of the basis and weight functions enables AES-FEM to overcome the element-quality dependence, while preserving the theoretical framework of FEM, without any complication in imposing essential boundary conditions or integrating the stiffness matrix. In addition, the resulting stiffness matrix of AES-FEM has virtually the same sparsity pattern as that of the classical FEM, while allowing the use of higher-degree polynomials and hence significantly improved accuracy.

As a general method, AES-FEM allows polynomial basis functions of arbitrary degrees. In this paper, we focus on the use of quadratic polynomial basis, for which the stiffness matrix has virtually the same sparsity pattern as the classical FEM with linear basis functions. However, AES-FEM is based on the more general weighted-residual formulation instead of the Galerkin formulation, and hence the resulting system is nonsymmetric, which is more expensive to solve than for symmetric matrices. In addition, it is more expensive to construct the basis functions of the AES-FEM than

to use the standard basis functions in FEM. Therefore, AES-FEM is conceivably less efficient than FEM for a given mesh. However, as we will demonstrate in our experimental results, AES-FEM is significantly more accurate than FEM on a given mesh due to its use of higher-degree basis functions, and it is also more efficient than FEM in terms of error versus runtime. Most impotently, AES-FEM enables better stability and faster convergence of iterative solvers than FEM over poor-quality meshes.

The reminder of the paper is organized as follows. In Section 2, we present some background information and recent developments of related methods. In Section 3, we formulate FEM from a weighted residual perspective, describe the construction of generalized Lagrange polynomial basis functions based on weighted least squares approximations, and then introduce AES-FEM. In Section 4, we analyze the consistency and stability of AES-FEM. In Section 5, we discuss some implementation details including the utilized data structure and the applicable algorithms. In Section 6, we present the results of some numerical experiments with our approach. Finally, Section 7 concludes the paper with a discussion.

2. BACKGROUND AND RELATED WORK

In this section, we review some background information and some methods closely related to our proposed approach, including the diffuse element method and element free Galerkin method, some modern variants or generalizations of FEM, as well as the generalized finite different methods.

2.1. Diffuse Element Method and Element Free Galerkin Method

Various alternatives of finite element methods have been proposed in the literature to mitigate the mesh-quality dependence. One of the examples is the *diffuse element method* (DEM) [10], proposed by Nayroles, Touzot, and Villon in 1992. Similar to AES-FEM, DEM constructs local approximations to an unknown function based on a local weighted least squares fitting at each node. However, unlike AES-FEM, the DEM is based on the Galerkin formulation, which requires the shape functions to have a compact support for efficiency. To this end, DEM relies on a weight function that vanishes at a certain distance from a node, in a manner similar to the moving least squares fittings [20]. The accuracy and efficiency of numerical integration in DEM depends on the particular choice of the weight function. In contrast, based on a weighted-residual formulation, AES-FEM enforces the compact support of the weak form with the weight functions, so the shape function does not need to have a compact support.

Another approach, which is closely related to DEM, is the *element-free Galerkin method* (EFGM) [9], proposed by Belytschko, Lu, and Gu in 1994. As a Galerkin method, EFGM also requires a compact support of its shape functions, which serve as both the trial functions and weight functions. Similar to DEM, EFGM constructs the shape functions based on moving least squares, for which the weight function plays a critical role in terms of accuracy and efficiency. However, depending on the weight functions, the shape functions in EFGM may not be polynomials. It requires special quadrature rules with more quadrature points than those of standard FEM [9], and it also requires evaluating a moving least squares fitting at each quadrature point. In addition, EFGM requires the

use of Lagrange multipliers for essential boundary conditions. In contrast, AES-FEM can utilize the same treatments of boundary conditions and numerical integration as the standard FEM.

2.2. Other Variants and Generalizations of FEM

Besides DEM and EFGM, various other generalizations of FEM have been developed in recent years. Some of the most notable ones include the generalized finite element method (GFEM) and extended finite element method (XFEM) [17, 21, 22, 23, 18]. These methods also alleviate the dependence on the mesh, by introducing enrichment functions to replace the standard FEM basis functions in regions with discontinuous solutions, such as along cracks. These enrichment functions in general are not polynomials, so special quadrature rules may be needed for efficiency. Away from the discontinuities, GFEM and XFEM rely on the standard FEM discretizations, so good mesh quality is still required in general.

The GFEM and XFEM may be viewed as special cases of the partition of unity method (PUM) [19], which is a general framework for constructing alternative basis functions. As noted in [19], the main outstanding questions of PUM include the choice of the basis functions, the imposition of essential boundary conditions, and the efficient numerical integration. The AES-FEM proposes a set of generalized Lagrange polynomial basis functions that also satisfy the partition of unity. Therefore, AES-FEM effectively addresses these open problems in PUM.

Besides the aforementioned generalizations of FEM, it is also worth noting the least-squares finite element method (LSFEM) [11]. LSFEM uses the concept of least squares globally to minimize a global error. In contrast, AES-FEM uses least squares in a local sense for constructing basis functions. To use LSFEM, any higher order PDE must be decomposed into a system of first order PDEs first, so it does not preserve the framework of the standard FEM.

Finally, we note the recent development of isogeometric analysis (IGA) [24], which uses NURBS (Non-Uniform Rational B-Splines) or T-splines as basis functions instead of the standard FEM basis functions. These methods can deliver high accuracy over very coarse meshes and can be advantageous for problems that can benefit from high-degree continuity, such as thin-shell modeling. However, IGA does not alleviate the dependency on mesh quality, since NURBS in effect impose stronger requirement on mesh quality than the standard FEM.

2.3. Generalized Finite Difference and Weighted Least Squares

The finite difference methods and finite element methods are closely related to each other. On structured meshes, the equivalence of these methods can be established in some special cases. While finite element methods were developed to support unstructured meshes from the beginning, the finite difference methods can also be generalized to unstructured meshes. These generalizations are often collectively referred to as *generalized finite difference* (or *GFD*) methods. The earlier GFD methods were based on polynomial interpolation; see e.g., [25], [12], and [26]. These methods construct a local multivariate polynomial interpolation by requiring the number of points in the stencil to be equal to the number of coefficients in the interpolant. However, due to the irregular distribution of points, the resulting Vandermonde matrices are often singular or ill-conditioned.

More general than an interpolant is the least squares or weighted least squares approximations, which allow more points in the stencil than the number of coefficients of a polynomial. Some earlier

examples of least-squares approximations include [13] and [14], which attempted to improve the conditioning of interpolation-based GFD. More recently, the least-squares-based GFD have been analyzed more systematically by Benito et al. [15, 27], and it been successfully applied to the solutions of parabolic and hyperbolic PDEs [15, 28, 27] and of advection-diffusion equation [29]. It has also been utilized in the weak-form of the Poisson equation, under the name *meshless finite difference method* [16, 30]. In these methods, a weighting scheme is often used to alter the norm that is being minimized, and hence the name *weighted least squares* (WLS). These weights serve a different role from those in the moving least squares [20], DEM, and EFGM. They do not need to have a compact support, and do not even to be defined by a continuous function in general.

Even though the least-squares approximations tend to be better conditioned than their interpolation-based counterparts, ill-conditioning may still occur for a given set of points. To overcome the issue, Jiao et al. utilized adaptive stencils coupled with column and row scaling, QR factorization with column pivoting, and a condition number estimator, which effectively guarantee the conditioning and hence the stability of approximations based on WLS [31, 32]. In AES-FEM, we extend this previous work to construct the generalized Lagrange polynomial basis functions, and then utilize these basis functions in the weak form of the finite element methods.

3. FORMULATION OF AES-FEM

The main idea of AES-FEM is the use of an alternative set of basis functions. For the basis functions, we propose the use of a set of generalized Lagrange polynomial basis functions (GLPBF) computed using a weighted least squares formulation. We use the standard FEM (hat) basis functions for the weight functions (a.k.a. test functions). In this section, we describe the weighted residual formulation of FEM, define generalized Lagrange polynomial basis functions based on weighted least squares, and then describe AES-FEM.

3.1. Weighted Residual Formulation of FEM

We will approach the formulation of the finite element method through the lens of the weighted residual method for solving PDEs. The main idea of the formulation is to consider the unknown solution as a linear combination of basis (trial) functions and then select the coefficients such that the residual is orthogonal to a set of weight (test) functions. Depending on the choice of the weight functions, one will derive different numerical methods, such as the collocation method, the least squares method, and the Galerkin method. Details about weighted residual and FEM can be found in [33, 34, 35]. In the following, we give a brief overview of weighted residuals for completeness.

Consider a linear differential operator $\mathcal{L}$ defined on a bounded, simply-connected domain Ω, with outward unit normal vector $\boldsymbol{n}$. Denote the boundary of Ω as $\Gamma = \Gamma_D \cup \Gamma_N$, where Γ_D and Γ_N are disjoint sets on which Dirichlet and Neumann boundary conditions are specified, respectively. We want to find a function u such that

$$\mathcal{L}u = f \tag{1}$$

subject to the boundary conditions

$$u = g \text{ on } \Gamma_D \quad \text{and} \quad \frac{\partial u}{\partial \boldsymbol{n}} = h \text{ on } \Gamma_N. \tag{2}$$

Eq. (1) is the strong form of the PDE. In the weighted residual formulation, we use the weak form based on a set of weight functions $\Psi = \{\psi_1, \ldots, \psi_n\}$, by requiring the residual $\mathcal{L}u - f$ to be orthogonal to ψ_i, i.e.,

$$\int_\Omega \psi_i \left(\mathcal{L}u - f \right) dV = 0. \tag{3}$$

To approximate u, let $\Phi = \{\phi_1, \ldots, \phi_n\}$ be a set of basis functions, and we define an approximation

$$u \approx \sum_{j=1}^n u_j \phi_j. \tag{4}$$

Substituting (4) into the weak form (3) and rearranging the equations, we then obtain

$$\sum_{j=1}^n u_j \int_\Omega \psi_i \left(\mathcal{L}\phi_j \right) dV = \int_\Omega \psi_i f \, dV. \tag{5}$$

At this point for simplicity, let us consider the Poisson equation with Dirichlet boundary conditions, for which the weak form is given by

$$\int_\Omega \psi_i \nabla^2 u \, dV = \int_\Omega \psi_i f \, dV. \tag{6}$$

Substituting (4) into (6), we obtain

$$\sum_{j=1}^n u_j \int_\Omega \psi_i \nabla^2 \phi_j \, dV = \int_\Omega \psi_i f \, dV. \tag{7}$$

The finite element method uses the integration by parts to reduce the order of derivatives required by (7). If ψ_i has weak derivatives and satisfies the condition $\psi_i|_{\Gamma_D} = 0$, then after integrating by parts and imposing the boundary conditions, we arrive at

$$-\sum_{j=1}^n u_j \int_\Omega \nabla \psi_i \cdot \nabla \phi_j \, dV = \int_\Omega \psi_i f \, dV. \tag{8}$$

Taking (8) over the n weight functions, we obtain the linear system

$$\boldsymbol{Ku} = \boldsymbol{g}, \tag{9}$$

where $\boldsymbol{K}$ is the stiffness matrix and $\boldsymbol{g}$ is the load vector, with

$$k_{ij} = -\int_\Omega \nabla \psi_i \cdot \nabla \phi_j \, dV \quad \text{and} \quad g_i = \int_\Omega \psi_i f \, dV. \tag{10}$$

If the weight functions are chosen to be the same as the basis functions, then we will arrive at the Galerkin method. In this paper, we introduce a new set of basis functions based on weighted least squares and we use the standard linear FEM "hat functions" as the weight functions.

3.2. Weighted Least Squares Approximations

In this subsection, we review numerical differentiation based on weighted least squares approximations, as described in [31, 32]. Similar to interpolation-based approximations, this method is based on Taylor series expansion. Let us take 2D as an example, and suppose $f(\boldsymbol{u})$ is a bivariate function with at least $d + 1$ continuous derivatives in some neighborhood of $\boldsymbol{u}_0 = (0, 0)$. Denote $c_{jk} = \frac{\partial^{j+k}}{\partial u^j \partial v^k} f(\boldsymbol{u}_0)$. Then for any $\boldsymbol{u}$ in the neighborhood, f may be approximated to the $(d + 1)$st order accuracy about the origin $\boldsymbol{u}_0$ as

$$f(\boldsymbol{u}) = \underbrace{\sum_{p=0}^{d} \sum_{\substack{j,k \geq 0}}^{j+k=p} c_{jk} \frac{u^j v^k}{j! k!}}_{\text{Taylor Polynomial}} + \underbrace{\mathcal{O}(\|\boldsymbol{u}\|^{d+1})}_{\text{remainder}}. \tag{11}$$

Analogous formulae exist in 1D and 3D. The derivatives of the Taylor polynomial are the same as those of f at $\boldsymbol{u}_0$ up to degree d. Therefore, once we have calculated the coefficients for the Taylor polynomial, finding the derivatives of f at $\boldsymbol{u}_0$ is trivial. We proceed with calculating the coefficients as follows.

For any point $\boldsymbol{u}_0$, we select a stencil of m nodes from the neighborhood around $\boldsymbol{u}_0$. Stencil selection is described further in Section 5. We do a local parameterization of the neighborhood so that $\boldsymbol{u}_0$ is located at the origin $(0, 0)$ and the coordinates of the other points are given relative to $\boldsymbol{u}_0$. Then substituting these points into (11), we obtain a set of approximate equations

$$\sum_{p=0}^{d} \sum_{\substack{j,k \geq 0}}^{j+k=p} c_{jk} \frac{u_i^j v_i^k}{j! k!} \approx f_i, \tag{12}$$

where $f_i = f(\boldsymbol{u}_i)$ and the c_{jk} denote the unknowns, resulting in an $m \times n$ system. There are $n = (d+1)(d+2)/2$ unknowns in 2D and $n = (d+1)(d+2)(d+3)/6$ unknowns in 3D. Let $\boldsymbol{V}$ denote the generalized Vandermonde matrix, $\boldsymbol{c}$ denote the vector of unknowns (i.e., the c_{jk}) and $\boldsymbol{f}$ denote the vector of function values. Then we arrive at the rectangular system

$$\boldsymbol{V} \boldsymbol{c} \approx \boldsymbol{f}. \tag{13}$$

Let us now introduce some notation to allow us to write the Taylor series in matrix notation before we proceed with our discussion of solving (13). Let $\boldsymbol{\mathcal{P}}_k^{(d)}(\boldsymbol{x})$ denote the set of all k-dimensional monomials of degree d and lower, stored in ascending order as a column vector. If no ambiguities will arise, we will use $\boldsymbol{\mathcal{P}}$ in place of $\boldsymbol{\mathcal{P}}_k^{(d)}(\boldsymbol{x})$. For example, for second degree in 2D we have

$$\boldsymbol{\mathcal{P}}_2^{(2)}(\boldsymbol{x}) = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 \end{bmatrix}^T. \tag{14}$$

Let $\boldsymbol{D}$ be a diagonal matrix consisting of the fractional factorial part of the coefficients, i.e. $\frac{1}{j!k!}$ in (11). For example, for second degree in 2D we have

$$\boldsymbol{D} = \text{diag}\left(1,\ 1,\ 1,\ \frac{1}{2},\ 1,\ \frac{1}{2}\right). \tag{15}$$

Then we may write the Taylor series as

$$f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{D} \mathcal{P}\left(\boldsymbol{x}\right). \tag{16}$$

To solve (13), we use a weighted linear least squares formulation [36], that is, we will minimize a weighted norm (or semi-norm)

$$\min_{\boldsymbol{c}} \|\boldsymbol{V}\boldsymbol{c} - \boldsymbol{f}\|_{\boldsymbol{W}} \equiv \min_{\boldsymbol{c}} \|\boldsymbol{W}\left(\boldsymbol{V}\boldsymbol{c} - \boldsymbol{f}\right)\|_2, \tag{17}$$

where $\boldsymbol{W}$ is an $m \times m$ diagonal weighting matrix. The entries of $\boldsymbol{W}$ assign weights to the rows of matrix $\boldsymbol{V}$. Specifically, if we denote the diagonal entries of $\boldsymbol{W}$ as w_i, then row i is assigned weight w_i. These weights can be used to prioritize the points in the system: we assign heavier weights to the nodes that are closer to the center point. By setting a weight to zero (or very close to zero), we may also filter out outliers or other undesirable points. Note that for a given node, the weighting matrix $\boldsymbol{W}$ is constant.

If $\boldsymbol{f}$ is in the column space of $\boldsymbol{V}$, then the solution of the linear system is not affected by a nonsingular weighting matrix. However, if $\boldsymbol{f}$ is not in the column space, which is often the case, then different weighting schemes can lead to different solutions. Choosing a good weighting matrix is application specific. For quadratic approximations, we compute the weights as follows. Let h denote the maximum radius of the neighborhood, that is

$$h = \max_{1 \le i \le m} \{\|\boldsymbol{u}_i\|_2\}. \tag{18}$$

Then

$$w_i = \left(\frac{\|\boldsymbol{u}_i\|_2}{h} + \epsilon\right)^{-1}, \tag{19}$$

where ϵ is a small number, such as $\epsilon = 0.01$, for avoiding division by zero.

After the weighting matrix has been applied, we can denote the new system as

$$\boldsymbol{M}\boldsymbol{c} \approx \tilde{\boldsymbol{f}}, \quad \text{where} \quad \boldsymbol{M} = \boldsymbol{W}\boldsymbol{V} \text{ and } \tilde{\boldsymbol{f}} = \boldsymbol{W}\boldsymbol{f}. \tag{20}$$

This resulting system may be rank-deficient or ill-conditioned. This is a challenge that GFD researchers have been dealing with since the 1970s [12]. The ill-conditioning may arise from a number of issues including poor scaling, an insufficient number of nodes in the neighborhood, or a degenerate arrangement of points. We resolve these issues with neighborhood selection, discussed in Section 5. We can address the scaling issue with the use of a diagonal scaling matrix $\boldsymbol{S}$. Let $\boldsymbol{a}_j$ denote the jth column of an arbitrary matrix $\boldsymbol{A}$. A typical choice for the jth entry of $\boldsymbol{S}$ is either $1/\|\boldsymbol{a}_j\|_2$, which approximately minimizes the 2-norm condition number of $\boldsymbol{A}\boldsymbol{S}$ [36], or $1/\|\boldsymbol{a}_j\|_\infty$ [37]. Using exact arithmetic, the matrix $\boldsymbol{S}$ does not affect the solution, but it can significantly improve the

conditioning and thus the accuracy in the presence of rounding errors. After applying the scaling matrix to $\boldsymbol{WV}$, the problem becomes

$$\min_{\boldsymbol{d}} \left\| \tilde{\boldsymbol{V}}\boldsymbol{d} - \tilde{\boldsymbol{f}} \right\|_2, \quad \text{where } \tilde{\boldsymbol{V}} \equiv \boldsymbol{WVS} = \boldsymbol{MS} \text{ and } \boldsymbol{d} \equiv \boldsymbol{S}^{-1}\boldsymbol{c}. \tag{21}$$

Conceptually, the solution to the above problem may be reached through the use of a pseudoinverse. We will have

$$\boldsymbol{d} = \tilde{\boldsymbol{V}}^+ \tilde{\boldsymbol{f}} \quad \text{where } \tilde{\boldsymbol{V}}^+ \equiv \left(\tilde{\boldsymbol{V}}^T \tilde{\boldsymbol{V}} \right)^{-1} \tilde{\boldsymbol{V}}^T. \tag{22}$$

However, since the resulting system may still be rank-deficient or ill-conditioned, we solve it using QR factorization with column pivoting, as discussed in Subsection 5.4. Finally, we get the vector of partial derivatives for the Taylor polynomial

$$\boldsymbol{c} = \boldsymbol{Sd}. \tag{23}$$

3.3. Generalized Lagrange Polynomial Basis Functions

We now define basis functions based on weighted least squares. Note that the standard finite element methods use piecewise Lagrange polynomial shape functions, which have two especially important properties: the coefficients of the basis functions have the physical meaning of the function values or their approximations at the nodes, and the basis functions form a partition of unity. We refer to the two properties as *function value as coefficient* and *partition of unity*, respectively. These properties are desirable in ensuring the consistency of the method based on these basis functions and also for the ease of imposing Dirichlet boundary conditions. However, the traditional concept of the Lagrange basis functions is interpolatory, so they are not applicable to least squares. We now generalize this concept, so that it can be applicable to least-squares-based basis functions.

Definition 1. Given a set of degree-d polynomial basis functions $\{\phi_i\}$, we say it is a set of degree-d *generalized Lagrange polynomial (GLP) basis functions* if:

1. $\sum_i f(x_i) \phi_i$ approximates a function f to $\mathcal{O}\left(h^{d+1}\right)$ in a neighborhood of the stencil, where h is some characteristic length measure, and
2. $\sum_i \phi_i = 1$.

We now define a set of GLP basis functions based on weighted least squares.

Given a stencil $\{x_i\}$, we follow the procedure in Subsection 3.2. When computing the jth basis function ϕ_j, let $\boldsymbol{f} = \boldsymbol{e}_j$, where $\boldsymbol{e}_j$ is the jth column of the identity matrix. Following (22) and (23), we have

$$\boldsymbol{c} = \boldsymbol{S}\tilde{\boldsymbol{V}}^+ \boldsymbol{W}\boldsymbol{e}_j. \tag{24}$$

Thus for the jth basis function, the vector $\boldsymbol{c}$ is exactly the jth column of $\boldsymbol{S}\tilde{\boldsymbol{V}}^+\boldsymbol{W}$. We define a set of basis functions as

$$\Phi = \left(\boldsymbol{S}\tilde{\boldsymbol{V}}^+\boldsymbol{W} \right)^T \boldsymbol{DP}. \tag{25}$$

For a more concrete example, if we denote the elements of $\tilde{V}^+$ as a_{ij} we can see that the jth basis function for degree 2 in 2D is

$$\phi_j = w_j \left(a_{1j}s_1 + a_{2j}s_2 x + a_{3j}s_3 y + a_{4j}s_4 \frac{1}{2}x^2 + a_{5j}s_5 xy + a_{6j}s_6 \frac{1}{2}y^2 \right). \qquad (26)$$

Note that w_j is a constant scalar, so ϕ_j is a polynomial. The basis functions in (25) are an example of GLP basis functions. We summarize this key feature in the following theorem.

Theorem 2. *The basis functions based on weighted least squares as defined in (25) are generalized Lagrange polynomial basis functions.*

We shall postpone the proof of this theorem to Section 4, where we will also analyze the accuracy and stability of finite element discretization based on these basis functions. In the following, we will finish the description of AES-FEM.

3.4. Description of AES-FEM

Starting with the weighted residual formulation for FEM from (8), we propose using GLP basis functions for the basis functions in the weak form and using the traditional hat functions for the weight functions. More specifically, for a given node i and its associated weight function ψ_i, a specific set of GLP basis functions $\{\phi_j\}$ is constructed from a weighted stencil (X_i, w_i) of n neighboring vertices centered at node i. This weight function ψ_i and its associated set of GLP basis functions $\{\phi_j\}$ are used to compute the ith row of the stiffness matrix, as given by (10). Note that, because a different set of basis functions is associated with each weight function, the basis functions on a given element differ row-to-row in the stiffness matrix.

For AES-FEM, when using the 1-ring neighborhood of a vertex as the stencil, we can use quadratic GLP basis functions, which have the advantage of decreased dependence on element quality and improved accuracy over standard finite element with linear basis functions, while virtually preserving the sparsity pattern of the stiffness matrix, as we will demonstrate in Section 6.

In terms of the computation of the load vector, we can use either the FEM or AES-FEM basis functions. We refer to the AES-FEM with these two options as "AES-FEM 1" and "AES-FEM 2," respectively. Additional implementation details will be given in Subsection 5.4.

4. ANALYSIS OF AES-FEM

In this section, we analyze the consistency and stability of AES-FEM. We start by explaining the applicability of Green's identity to the GLP basis functions. We will then prove that basis functions in (25) are GLP basis functions and discuss the consistency and stability of AES-FEM.

4.1. Green's Identity and Integration by Parts

For a given weight function ψ, the GLP basis functions are continuously differentiable everywhere in within the domain of integration. Hence, in a *variational sense*, the GLP basis functions still allow one to use Green's identities to formulate a weak form. Let ϕ_j be any GLP basis function

computed from a weighted stencil, and ψ_i be a classical FEM shape function with compact support contained within the set Ω. Therefore, for any partial derivative operation ∂, it follows that

$$\int_\Omega (\partial \phi_j)\,\psi_i\,\mathrm{d}x = -\int_\Omega \phi_j\,(\partial \psi_i)\,\mathrm{d}x. \tag{27}$$

Because of this property, the weak-form formulation with the GLP basis functions in AES-FEM is mathematically sound.

In addition, since the finite-element basis functions ψ_i vanish along the boundary, after integration by parts, we have the identities

$$\int_\Omega \psi_i \nabla^2 \phi_j\,dV = -\int_\Omega \nabla \psi_i \cdot \nabla \phi_j\,dV. \tag{28}$$

Therefore, we can reduce the order of derivatives similar to the classical FEM, without introducing additional boundary integrals to the computation.

4.2. Generalized Lagrange Polynomial Basis Functions

We now show that the basis functions in (25) are indeed generalized Lagrange polynomial basis functions, which follows from the two lemmas below.

Lemma 3. *Let $\{x_i\}$ be a stencil with m nodes and stencil diameter h. Let $\{\phi_j\}$ be the complete set of basis functions of degree up to d as defined by (25) on this stencil. Given an arbitrary function f, define the following approximation of f*

$$f^h(x) \equiv \sum_{j=1}^m f_j \phi_j(x) = \boldsymbol{f}^T \boldsymbol{\Phi}, \tag{29}$$

where $f_j = f(x_j)$ and $\boldsymbol{f} = [f_1\ f_2\ \ldots\ f_m]^T$. If the rescaled matrix $\tilde{V}$ has a bounded condition number, then f^h approximates f to $\mathcal{O}\left(h^{d+1}\right)$. In addition, given any degree-k differential operator $\mathcal{D}$, if f is continuously differentiable up to degree k, then $\mathcal{D}f^h$ approximates $\mathcal{D}f$ to $\mathcal{O}\left(h^{d-k+1}\right)$.

Proof

First, we show that the approximation f^h is equivalent to directly solving a weighted least squares problem for a local polynomial fitting of f. Using the method described in Subsection 3.2, we get the coefficients $\boldsymbol{c} = \boldsymbol{S}\tilde{\boldsymbol{V}}^+\boldsymbol{W}\boldsymbol{g}$ where $\boldsymbol{g} = [f_1\ f_2\ \ldots\ f_m]^T$. Thus, the local polynomial fitting of f is

$$\begin{aligned}
f(\boldsymbol{x}) &\approx \boldsymbol{c}^T \boldsymbol{D}\boldsymbol{\mathcal{P}} \\
&= \left(\boldsymbol{S}\tilde{\boldsymbol{V}}^+\boldsymbol{W}\boldsymbol{g}\right)^T \boldsymbol{D}\boldsymbol{\mathcal{P}} \\
&= \boldsymbol{g}^T \left(\boldsymbol{S}\tilde{\boldsymbol{V}}^+\boldsymbol{W}\right)^T \boldsymbol{D}\boldsymbol{\mathcal{P}} \\
&= \boldsymbol{g}^T \boldsymbol{\Phi} \\
&= f^h(\boldsymbol{x}).
\end{aligned}$$

It follows from Theorem 4 in [31] that f^h approximates f to $\mathcal{O}\left(h^{d+1}\right)$, and $\mathcal{D}f^h$ approximates $\mathcal{D}f$ to $\mathcal{O}\left(h^{d-k+1}\right)$ for any degree-k differential operator $\mathcal{D}$.   □

Lemma 4. *The basis functions in (25) form a partition of unity, i.e.,* $\sum_{j=1}^{m} \phi_j(x) = 1$.

Proof

We will show that on a given stencil $\{x_i\}$, the GLP basis functions $\{\phi_j\}$ form a partition of unity. Let V be the generalized Vandermonde matrix for the given stencil. For example, for second order expansion in 2D, we have

$$V = \begin{bmatrix} 1 & x_1 & y_1 & \cdots & \frac{1}{2}y_1^2 \\ 1 & x_2 & y_2 & \cdots & \frac{1}{2}y_2^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & y_m & \cdots & \frac{1}{2}y_m^2 \end{bmatrix}. \tag{30}$$

For a given function f, using the truncated Taylor series, we have

$$V c = g, \tag{31}$$

where c is the vector of partial derivative values. Applying the diagonal row weighting matrix W and the diagonal column scaling matrix S, we have

$$\tilde{V}\left(S^{-1}c\right) = Wg \quad \text{where} \quad \tilde{V} \equiv WVS. \tag{32}$$

This is a least squares problem, and the solution for c may be reached through the use of a pseudoinverse,

$$c = S\tilde{V}^{+}Wg. \tag{33}$$

For the jth GLP basis function, we have $g_j = [0 \ldots 1 \ldots 0]^T$, where the 1 is in the jth position, and hence the columns of $S\tilde{V}^{+}W$ multiplied by the Taylor constants D give the coefficients for the GLP basis functions. This implies that the entries of the ith row of $S\tilde{V}^{+}W$ correspond to the coefficients of the ith terms in the set of basis functions.

   To finish the proof, it suffices to show that the sum of the entries in the first row of $S\tilde{V}^{+}W$ is 1, and the sum of the entries in any other row is 0. Let vector w be the diagonal entries of W. Every entry of the first column of V is equal to 1, thus the first column of $\tilde{V}$ is then $s_1 w$. Denote the ith row of $\tilde{V}^{+}$ as $\tilde{v}_{(i,:)}^{+}$. The sum of the entries of the ith row of $S\tilde{V}^{+}W$ is $s_1 \tilde{v}_{(i,:)}^{+T} w$. Since $\tilde{V}^{+}$ is a left inverse of $\tilde{V}$, we have $\tilde{V}^{+}\tilde{V} = I$, and hence

$$s_1 \tilde{v}_{(i,:)}^{+T} w = \begin{cases} 1 & i = 1 \\ 0 & 2 \leq i \leq n \end{cases}. \tag{34}$$

Therefore, the GLP basis functions form a partition of unity.   □

   From the above lemmas, the basis functions in (25) satisfy both the properties of function value as coefficient and partition of unity, and hence they are GLP basis functions, as claimed in Theorem 2.

4.3. Consistency of AES-FEM

The accuracy of the AES-FEM depends on its consistency and stability. We first consider the consistency of AES-FEM, in terms of its truncation errors in approximating the weak form (3) by (5) using the GLP basis functions. In a nutshell, the consistency of the AES-FEM follows directly from Lemma 3. For completeness, we consider a specific example of solving the Poisson equation. The analysis for other PDEs can be derived in a similar fashion.

Theorem 5. *Suppose u is smooth and thus $\|\nabla u\|$ is bounded. Then, when solving the Poisson equation using AES-FEM with degree-d GLP basis functions in (8), for each ψ_i the weak form (6) is approximated to $\mathcal{O}(h^d)$, where h is some characteristic length measure of the mesh.*

Proof

Let u be the exact solution on a mesh with mesh size h, and let

$$\tilde{u} = \sum_{j=1}^{n} u_i \phi_j$$

denote the approximation to u using degree-d GLP basis functions. Because the test functions ψ_i vanishes along boundary, the weak form (6) can be rewritten as

$$-\int_{\Omega} \nabla u \cdot \nabla \psi_i \, dV = \int_{\Omega} \psi_i f \, dV.$$

When using degree-d GLP basis functions, it follows from Lemma 3 that

$$\|\nabla u - \nabla \tilde{u}\| = \mathcal{O}\left(h^d\right)$$

within each element. Under the assumption that u is twice differentiable, ∇u is bounded, and hence

$$\left| \int_{\Omega} (\nabla u - \nabla \tilde{u}) \cdot \nabla \psi_i \, dV \right| = \mathcal{O}\left(h^d\right).$$

Furthermore, if f is approximated by degree-d GLP basis functions as

$$\tilde{f} = \sum_{j=1}^{n} f_i \phi_j,$$

then we have

$$\left| \int_{\Omega} \left(f - \tilde{f}\right) \psi_i \, dV \right| = \mathcal{O}\left(h^{d+1}\right).$$

$\square$

More specifically, when using quadratic GLP basis functions, the truncation errors are second order in the stiffness matrix. The truncation errors in the load vector is third order for AES-FEM 2 when f is also approximated using quadratic GLP basis functions, but it is second order for AES-FEM 1 when f is approximated using linear FEM hat functions. Like most other PDE methods, as long as the method is stable, the rounding errors do not dominate the truncation errors, and there is

no systematic cancelation of truncation errors, we expect the solution to converge at the same rate as the local truncation errors, as we will demonstrate numerically in Section 6.

4.4. Stability

For elliptic PDEs, the stability of a method depends on the condition number of its coefficient matrix, which can affect the performance of iterative solvers and the accuracy of the solution. It is well known that the traditional finite element method may be unstable for poorly shaped meshes [38], and some meshless methods may also suffer from instability when two points nearly coincide. AES-FEM avoids these potential instability issues.

As a concrete example, let us consider the Poisson equations with Dirichlet boundary conditions, whose coefficient matrix is the stiffness matrix. It is well known that the condition number of the stiffness matrix is proportional to h^{-2}, where h is some characteristic length of the mesh [39]. However, if the condition number is significantly larger, then the method is said to be *unstable*, which can happen due to various reasons.

The ill-conditioning of any local stiffness matrix may lead to poor scaling and in turn ill-conditioning of the global stiffness matrix. This is owing to the following fact, which is given as Theorem 2.2.26 in [40].

Proposition 6. *For any matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$, its condition number in any p-norm, denoted by $\kappa_p(A)$, is bounded by the ratio of the largest and smallest column vectors in p-norm, i.e.,*

$$\kappa_p(A) \geq \frac{\max_{1 \leq i \leq n} \|a_i\|_p}{\min_{1 \leq j \leq n} \|a_j\|_p}, \tag{35}$$

where a_k denotes the kth column vector of A.

The above fact offers an intuitive explanation of a source of ill-conditioning in traditional finite element methods due to poorly shaped elements: poorly shaped elements may lead to unbounded large entries in local stiffness matrices, so the column norms of the global stiffness matrix would vary substantially, and in turn the global stiffness matrix is necessarily ill-conditioned. In the context of AES-FEM, there can be two potential sources of local instability due to poor scaling. First, the unnormalized local Vandermonde system given in (13) is in general very poorly scaled. We resolved this by normalizing the Vandermonde system to avoid poor scaling. Second, the normalized Vandermonde system may still be ill-conditioned occasionally, when a stencil is degenerate or nearly degenerate, which could lead to unbounded large values in the local stiffness matrix. We resolved this issue by using QR with column pivoting and condition-number estimation.

Even if the local stiffness matrices are bounded, the global stiffness matrix may still be ill-conditioned due to linearly dependent rows or columns. This is the potential source of instability for some meshless methods when two points nearly coincide; the two points may share the same stencil and basis functions, so that the rows or columns corresponding to the two points would be nearly identical. Therefore, these meshless methods also require good point distributions. In AES-FEM, we utilize the mesh topology to construct the stencil, as we will describe in Section 5. This ensures that no two vertices share the same stencil unless there are coincident points, and hence it gives a strong guarantee that the rows in the global stiffness matrix are linearly independent.
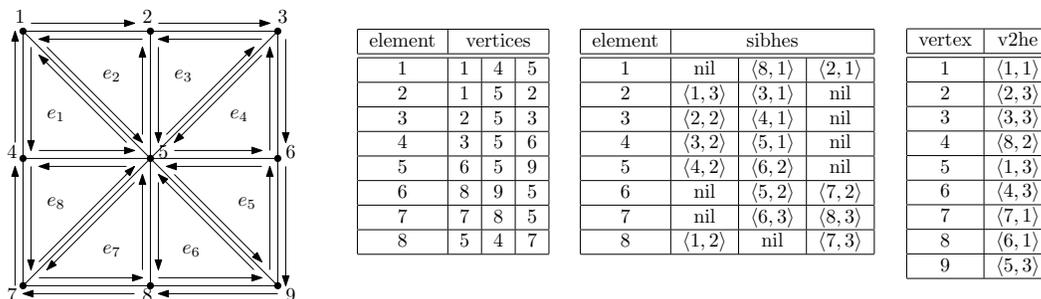
element	vertices		
1	1	4	5
2	1	5	2
3	2	5	3
4	3	5	6
5	6	5	9
6	8	9	5
7	7	8	5
8	5	4	7

element	sibhes		
1	nil	$\langle 8,1 \rangle$	$\langle 2,1 \rangle$
2	$\langle 1,3 \rangle$	$\langle 3,1 \rangle$	nil
3	$\langle 2,2 \rangle$	$\langle 4,1 \rangle$	nil
4	$\langle 3,2 \rangle$	$\langle 5,1 \rangle$	nil
5	$\langle 4,2 \rangle$	$\langle 6,2 \rangle$	nil
6	nil	$\langle 5,2 \rangle$	$\langle 7,2 \rangle$
7	nil	$\langle 6,3 \rangle$	$\langle 8,3 \rangle$
8	$\langle 1,2 \rangle$	nil	$\langle 7,3 \rangle$

vertex	v2he
1	$\langle 1,1 \rangle$
2	$\langle 2,3 \rangle$
3	$\langle 3,3 \rangle$
4	$\langle 8,2 \rangle$
5	$\langle 1,3 \rangle$
6	$\langle 4,3 \rangle$
7	$\langle 7,1 \rangle$
8	$\langle 6,1 \rangle$
9	$\langle 5,3 \rangle$

Figure 1. An example of half edges and associated data structure.

The aforementioned reasons are the most common causes of instability for solving elliptic PDEs. Another source of instability is a cluster of coincident points or inverted elements, which rarely happen in practice, and hence we defer their treatments to future work. As we will demonstrate numerically in Section 6, by resolving these instabilities, AES-FEM produces well-conditioned stiffness matrices for meshes, even with very bad quality elements or point distributions.

5. IMPLEMENTATION

We discuss the practical aspects of the implementation of AES-FEM in this section. We start with a discussion of the utilized mesh data structure and then explain how this enables quick and efficient neighborhood selection. Finally, the algorithms are presented and runtime is analyzed.

5.1. Data Structure

We use an Array-based Half-Facet (AHF) data structure [41] to store the mesh information. In a d-dimensional mesh, the term *facet* refers to the $(d-1)$-dimensional mesh entities; that is, in 2D the facets are the edges, and in 3D the facets are the faces. The basis for the half-facet data structure is the idea that every facet in a manifold mesh is made of two half-facets oriented in opposite directions. We refer to these two half-facets as *sibling half-facets*. Half-facets on the boundary of the domain have no siblings. In 2D and 3D, the half-facets are *half-edges* and *half-faces*, respectively. We identify each half-facet by a two tuple: the element ID and a local facet ID within the element. In 2D, we store the element connectivity, sibling half-edges, and a mapping from each node to an incident half-edge. In 3D, we store the element connectivity, sibling half-faces, and a mapping from each node to an incident half-face. For an example of a 2D mesh and the associated data structure, see Figure 1. This data structure allows us to do neighborhood queries for a node in constant time (provided the valance is bounded). For additional information about the AHF data structure, see [41].

5.2. Neighborhood Selection

The use of the AHF data structure allows us to quickly find the neighborhood of a node. We use the concept of rings to control the size of the neighborhood. The *1-ring neighbor elements* of a node
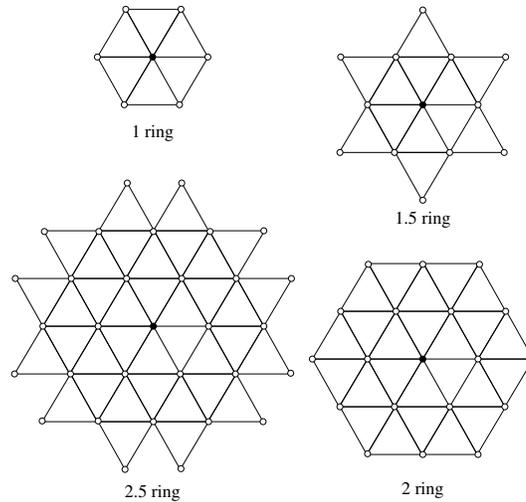
Figure 2. Examples of 2D stencils with 1-ring, 1.5-ring, 2-ring, and 2.5-ring neighborhoods of center node (in solid black).

are defined to be the elements incident on the node. The *1-ring neighborhood* of a node contains the nodes of its 1-ring neighbor elements [31]. Most of the time, when using GFD with second order basis functions or when constructing second order GLP basis functions, the 1-ring neighborhood of a node supplies the appropriate number of nodes. If the valance is low, it might be necessary to further expand and collect more nodes for the neighborhood. Therefore, for any integer $k \geq 1$, we define the $(k+1)$-*ring neighborhood* as the nodes in the k-ring neighborhood plus their 1-ring neighborhoods.

As k increases, the average size of the k-ring neighborhood grows very quickly. The granularity can be fine-tuned by using fractional rings. In 2D we use half rings, which are defined in [31]; for any integer $k \geq 1$ the $(k+1/2)$-*ring neighborhood* is the k-ring neighborhood plus the nodes of all the faces that share an edge with the k-ring neighborhood. See Figure 2 for a visualization of rings and half-rings in 2D. We extend this definition to 3D and introduce $1/3$- and $2/3$-rings. For any integer $k \geq 1$, the $(k+1/3)$-*ring neighborhood* contains the k-ring neighborhood plus the nodes of all elements that share a face with the k-ring neighborhood. The $(k+2/3)$-*ring neighborhood* contains the k-ring neighborhood plus the nodes of all faces that share an edge with the k-ring neighborhood.

Note that for 2D triangular and 3D tetrahedral meshes, the 1-ring neighborhood typically has enough points for constructing quadratic GLP basis functions. Therefore, the stiffness matrix from AES-FEM has a similar sparsity pattern to that from standard FEM with linear shape functions. However, when the 1-ring neighborhood is too small, the extended stencil with a larger ring allows AES-FEM to overcome mesh-quality dependence and improve its local stability.

5.3. Stable Computation of GLP Basis Functions

Even with proper neighborhood selection, it cannot be guaranteed that the least-squares problem in (21) will be well-conditioned. Hence, it is critical to use a robust method that ensures the accuracy and stability of the approximate solutions.

Algorithm 1 Initialization of a Generalized Vandermonde Matrix

---

function: initiate_GVM
input: 1. $\boldsymbol{x}_k$: local coordinates of stencil
        2. $\boldsymbol{w}$: vector of row weights
        3. p: desired degree for $\boldsymbol{V}$
        4. ϵ: tolerance for rank deficiency
output: struct gvm: with fields $\boldsymbol{W}, \boldsymbol{S}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{P}, r$ (estimated rank)

  1: create generalized Vandermonde matrix $\boldsymbol{V}$ from local coordinates $\boldsymbol{x}_k$
  2: determine column scaling matrix $\boldsymbol{S}$
  3: $\boldsymbol{W} \leftarrow \operatorname{diag}(\boldsymbol{w})$
  4: $\tilde{\boldsymbol{V}} \leftarrow \boldsymbol{W}\boldsymbol{V}\boldsymbol{S}$
  5: solve $\tilde{\boldsymbol{V}}\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{R}$
  6: estimate rank r from $\boldsymbol{R}$ so that $r = \max\left\{i | \operatorname{cond}\left(\boldsymbol{R}_{1:i,1:i}\right) \leq 1/\epsilon\right\}$

Note that one standard technique in linear algebra for solving rank-deficient least-squares problems is truncated singular value decomposition (TSVD) [36]. The TSVD is not recommended here, because it can result in the loss of partition of unity of the basis functions. The reason is as follows. When truncating SVD, one truncates any singular value σ_j that is smaller than $\epsilon\sigma_1$, where σ_1 is the largest singular value and ϵ is some small positive value, such as 10^{-4}. These singular values may be necessary for computing the constant terms in the GLP basis functions, and hence their loss can result in a set of basis functions that lack the partition of unity and in turn may compromise convergence.

We avoid the above issue by using truncated QR factorization with column pivoting (QRCP). When performing QRCP, one can find a numerical rank r of matrix $\boldsymbol{R}$ so that the condition number $\kappa(\boldsymbol{R}_{1:r,1:r}) < 1/\epsilon$, where ϵ is some small positive value, such as 10^{-4}. This is elaborated in the below discussion of Algorithm 1. If r is less than the size of $\boldsymbol{R}$, then any diagonal entry of $\boldsymbol{R}$ in position $r + 1$ or greater is truncated, thus truncating the $(r + 1)$th and subsequent columns of $\boldsymbol{Q}$. In QRCP, we require the first column not to be permuted, and this ensures the resulting basis functions satisfy the property of partition of unity.

In Algorithm 1, we present the procedure for initializing the generalized Vandermonde matrix $\tilde{\boldsymbol{V}}$ and factoring it using QRCP. The generalized Vandermonde matrix is formed from the local coordinates of the stencils and is scaled by the column scaling matrix $\boldsymbol{S}$ and the row scaling matrix $\boldsymbol{W}$. The resulting matrix is then factored using QRCP. We use a variant of Householder triangularization [36] since this procedure is more efficient and stable than alternatives (such as Gram-Schmidt orthogonalization). When implementing this procedure, the QR factorization of $\tilde{\boldsymbol{V}}$ can overwrite $\boldsymbol{V}$. The jth Householder reflection vector is of size $n - j + 1$. By requiring the first element of the vector to be positive, the first element may be reconstructed from the other elements, and thus only $n - j$ entries are required to store the jth Householder reflection vector. The Householder vectors are stored in the lower triangular part of $\boldsymbol{V}$ and the $\boldsymbol{R}$ entries are stored in the upper part. The permutation matrix $\boldsymbol{P}$ is stored in a permutation vector.

In addition to computing the QR factorization of the generalized Vandermonde matrix, an estimation of the numerical rank of $\boldsymbol{R}$ is also computed in Algorithm 1. The rank is important for ensuring the overall stability of other algorithms that use this initialization step. In order to estimate the numerical rank, we estimate the condition numbers of the leading principal sub-matrices of $\boldsymbol{R}$, $\boldsymbol{R}_{1:r,1:r}$, and find the largest r such that $\tilde{\kappa}(\boldsymbol{R}_{1:r,1:r}) \leq 1/\epsilon$ where $\tilde{\kappa}$ is the estimated condition number

---

Algorithm 2 Approximating $\mathcal{D}f$ at given point $\boldsymbol{x}$ from WLS

---

function diff_WLS
input: 1. struct gvm: with fields $\boldsymbol{W}$, $\boldsymbol{S}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{P}$, r (estimated rank)
    2. coefficients $\boldsymbol{a} = \mathcal{D}\mathcal{P}(\boldsymbol{x})$
output: weights $\boldsymbol{d}$, so that $\boldsymbol{d}^T \boldsymbol{g} = \mathcal{D}f(\boldsymbol{x})$ for $\boldsymbol{g}$ containing $f(\boldsymbol{x}_k)$ at stencil points
  1: $\boldsymbol{a} \leftarrow \left(\boldsymbol{P}_{:,1:r}\right)^T \boldsymbol{S}^{-1}\boldsymbol{a}$;
  2: $\boldsymbol{a} \leftarrow \boldsymbol{R}_{1:r,1:r}^{-T}\boldsymbol{a}$;
  3: $\boldsymbol{d} \leftarrow \boldsymbol{W}\boldsymbol{Q}_{:,1:r}\boldsymbol{a}$;

---

and ϵ is some given drop-off tolerance depending on the degree of polynomials. Note that since the matrix $\boldsymbol{R}$ is small, the condition numbers in different norms differ by only a small factor. Therefore for efficiency, we estimate the condition number of $\boldsymbol{R}$ in the 1-norm using the algorithm described in [42].

Once the generalized Vandermonde matrix has been initialized, it may be used to construct generalized finite differentiation operators from the weighted least squares approximations, as described in Algorithm 2. The input for this algorithm is the output from Algorithm 1 and a vector $\boldsymbol{a}$. The vector $\boldsymbol{a} = \mathcal{D}\mathcal{P}(\boldsymbol{x})$ contains the values for some specified derivative $\mathcal{D}$ of the monomial basis functions $\mathcal{P}$ at point $\boldsymbol{x}$. For example, let $\mathcal{D}$ be $\frac{\partial}{\partial y}$. Then in 2D, we have $\boldsymbol{a} = \mathcal{D}\mathcal{P}(x, y) = [0\ 0\ 1\ 0\ x\ 2y]^T$ and in 3D, we have $\boldsymbol{a} = \mathcal{D}\mathcal{P}(x, y, z) = [0\ 0\ 1\ 0\ 0\ x\ 0\ 2y\ 0\ 0]^T$. The algorithm returns a vector of weights $\boldsymbol{d}$ so that $\boldsymbol{d}^T \boldsymbol{g} = \mathcal{D}f(\boldsymbol{x})$ for a vector $\boldsymbol{g} = [f_1\ f_2\ \ldots\ f_m]^T$ containing the values of the function at the stencil points. Note that for a GLP basis function, the returned weights are the values of the specified derivative at the points in the stencil.

In terms of the computational cost, the step that dominates Algorithm 1 is the QRCP factorization, which takes $\mathcal{O}\left(2mn^2 - \frac{2}{3}n^3\right)$ flops where $\tilde{\boldsymbol{V}}$ is $m \times n$ [36]. Here, m is the number of points in the stencil and n is the number of terms in the Taylor series expansion (for second order expansion $n = 6$ in 2D and $n = 10$ in 3D). As long as the valance is bounded, that is m is bounded, this algorithm is executed in a constant time. Compared to calculations based on the standard finite-element basis functions, which are tabulated, the computation based on the GLP basis functions is more expensive. This leads to higher cost of AES-FEM in assembling the stiffness matrix and load vector, as we discuss next. However, this cost is a small constant per element, and AES-FEM can be more efficient overall by delivering higher accuracy, as we will demonstrate in Section 6.

5.4. Assembly of Stiffness Matrix and Load Vector

Algorithm 3 presents a summary of the AES-FEM procedure for assembling the stiffness matrix and load vector for a PDE with Dirichlet boundary conditions. Unlike the standard FEM procedure, we build the stiffness matrix row by row, rather than element by element. This is because the most computationally expensive part of the procedure is to compute the derivatives for the set of basis functions on each stencil. A weight function is nonzero only on the neighborhood around its corresponding node. Since the weight functions correspond to the rows, we assemble the stiffness matrix row by row, ensuring that we will only need to compute the derivatives for each neighborhood once.

When computing a row of the stiffness matrix, the first step is to obtain the stencil of node k. This step is performed by utilizing the data structure presented in Subsection 5.1 and the proper

size of the stencil is ensured by choosing the ring sizes adaptively. Next, the local coordinates are calculated for the points in the stencils and the row weights are computed. Using Algorithm 1, the QR factorization of the generalized Vandermonde matrix is computed for the neighborhood. Then for each element that contains node k, we perform the integration of the weak form in a manner that is similar to standard FEM. The element Jacobian is computed and used to find the local coordinates of the quadrature points. The derivatives of the weight function are computed, that is $\nabla \psi_i$, at the quadrature points of the current element. Recall that the weight functions are the standard hat functions. Next the derivatives of the basis functions, that is $\nabla \phi_j$, are computed at the quadrature points of the current element. The basis functions are the GLP basis functions and thus Algorithm 2 is used. The value of the integral on the current element is computed and either added to the stiffness matrix or subtracted from the load vector, depending on whether the basis function corresponds to a node with Dirichlet boundary conditions.

When computing the load vector, typically the entries $b_i = \int \psi_i f \ dV$ are computed using a quadrature rule. One may evaluate f at the quadrature points in two ways. The first way is to use the standard procedure in FEM, i.e., to use the FEM basis functions and the values of f at the nodes of the element. Let M be the vector of FEM shape functions evaluated at quadrature point x_k and g_{elem} be the vector of the function values at the nodes of the element. Then, we have

$$f\left(x_k\right) = M \cdot g_{\mathrm{elem}}. \tag{36}$$

Alternatively, we may use GLP basis functions to interpolate the values of f at the quadrature points. We can approximate an arbitrary function using the set of GLP basis functions. In matrix notation, we have

$$f\left(x_k\right) = g_{\mathrm{sten}}^T \left(S\tilde{V}^+ W\right)^T D\mathcal{P}\left(x_k\right), \tag{37}$$

where g_{sten} is the vector of function values at the nodes in the stencil and the vector $\mathcal{P}\left(x_k\right)$ has been evaluated at the quadrature point x_k. As mentioned earlier, we refer to the variant of AES-FEM using the former method of calculating the load vector as AES-FEM 1 and refer to the latter variant as AES-FEM 2.

We use Gaussian quadrature to perform the integration within each element. For quadratic GLP basis functions in 2D, we use a 1-point rule for stiffness matrix, and a 3-point rule for the load vector. In 3D, we use a 1-point rule for the stiffness matrix and a 4-point rule for the load vector. These rules are exact because the basis functions and their derivatives are quadratic and linear, respectively.

It is worth noting that because of the properties of generalized Lagrange polynomial basis functions, Dirichlet boundary conditions may be imposed in AES-FEM in the same manner as in standard FEM. One does not need to use Lagrange multipliers or a penalty method. Additionally, the standard method for imposing Neumann boundary conditions may be used in AES-FEM.

All the steps inside of the primary for-loop are executed in constant time, assuming that the size of each neighborhood is bounded. Therefore, the assembly of the stiffness matrix in AES-FEM has an asymptotic runtime of $\mathcal{O}(n)$, where n is the number of nodes in the mesh. When using AES-FEM 2, that is when WLS approximation is used to compute the approximation of f at the quadrature points, Algorithm 2 is called again. While this function is constant in runtime, it has a large coefficient and thus takes longer than approximating the values of f using FEM (hat) basis functions. Therefore, the assembly time for AES-FEM 2 is longer than that for AES-FEM 1, as can be seen in Section 6.

Algorithm 3 Building a Stiffness Matrix and Load Vector using AES-FEM

---

function: aes_fem
input: 1. x, `elem`, `opphfs`, `vh2f`: mesh information
        2. p: desired degree for GLP functions
        3. ϵ: tolerance for rank deficiency
        4. `AESFEM1`: boolean for AES-FEM 1 or AES-FEM 2
        5. `isDBC`: flags for Dirichlet boundary conditions
output: stiffness matrix K and load vector b

1: **for** each node without Dirichlet boundary conditions **do**
2:    obtain neighborhood of node
3:    calculate local parameterization x_k and row weights w for neighborhood
4:    aes_gvm $\leftarrow$ initiate_GVM(x_k, w, p, ϵ)
5:    obtain local element neighborhood
6:    **for** each element in local neighborhood **do**
7:       calculate element Jacobian and local coordinates of quad-points
8:       calculate derivatives of FEM shape functions at quad-points
9:       $a \leftarrow \mathcal{DP}(x)$ where $\mathcal{DP}(x)$ is defined by the PDE we are solving
10:      GLPderivs $\leftarrow$ diff_WLS(aes_gvm, a)
11:      **for** each node in neighborhood **do**
12:        **if** not Dirichlet BC node **then**
13:          add integral to appropriate stiffness matrix entry
14:        **else**
15:          subtract integral from load vector
16:        **end if**
17:      **end for**
18:      **if** `AESFEM1` **then**
19:        calculate load vector over current element using FEM approximations for quad-points
20:      **else**
21:        calculate load vector over current element using GLP approximations for quad-points
22:      **end if**
23:    **end for**
24: **end for**

Once the stiffness matrix and the load vector are assembled, we use the generalized minimal residual method (GMRES) [43] with a preconditioner to solve the linear system. GMRES is a standard Krylov subspace method for iteratively solving a sparse, nonsymmetric linear system. Specifically, we use the MATLAB implementation of GMRES. Another option for solving this system would be multigrid methods [44], and we will explore their use in future work. As a preconditioner, we use incomplete LU factorization (ILU) in 2D and Gauss-Seidel in 3D.

Finally, for completeness, we include Algorithm 4 to summarize the GFD procedure for solving PDEs with Dirichlet boundary conditions. We use this algorithm primarily for comparison with the AES-FEM algorithm. We can see that for GFD, we need to use Algorithm 2 to compute the weights once for each non-Dirichlet node in the mesh; that is, if there are n non-Dirichlet nodes, Algorithm 2 is called n times. For AES-FEM, for a given node, we use this algorithm once for every element containing that node. Thus if every node has a neighborhood of k elements and there are n non-Dirichlet nodes, we call the algorithm kn times. Therefore, the assembly time is longer for AES-FEM than for GFD, as we will see in Section 6.

Algorithm 4 Constructing a GFD coefficient matrix

---

function: gfd
input: 1. x, `elem`, `opphfs`, `vh2f`: mesh information
        2. p: desired degree for GFD functions
        3. `isDBC`: flags for Dirichlet boundary conditions
output: GFD matrix K and vector b

 1: **for** each node without Dirichlet boundary conditions **do**
 2:    obtain neighborhood of node
 3:    calculate local parameterization and row weights for neighborhood
 4:    gfd_cvm $\leftarrow$ initiate_CVM(x_k, w, p, ϵ)
 5:    $a \leftarrow \mathcal{DP}(x)$ where $\mathcal{DP}(x)$ is defined by the PDE we are solving
 6:    GFDderivs $\leftarrow$ diff_WLS(gfd_cvm, a)
 7:    **for** each node in local neighborhood **do**
 8:      **if** not BC node **then**
 9:        enter value in matrix
10:      **else**
11:        subtract from RHS vector
12:      **end if**
13:    **end for**
14: **end for**

---

6. NUMERICAL RESULTS

In this section, we compare the accuracy, efficiency, and element shape quality dependence of AES-FEM 1, AES-FEM 2, FEM with linear basis functions, and GFD with quadratic basis functions. We compare these four method because the sparsity pattern of the coefficient matrix is nearly identical for all the methods. The sparsity pattern determines the amount of storage necessary and also the computational cost of vector-matrix multiplication. The errors are calculated using the discrete L_2 and L_∞ norms. Let u denote the exact solution and let $\hat{u}$ denote the numerical solution. Then, we calculate the norms as

$$L_2\,(\text{error}) = \left(\int_\Omega |\hat{u} - u|^2 \partial\Omega \right)^{1/2} \quad \text{and} \quad L_\infty\,(\text{error}) = \max_i |\hat{u} - u|. \tag{38}$$

For each series of meshes of different grid resolution, we calculate the average convergence rate as

$$\text{convergence rate } = -\log_2\left(\frac{\text{error of mesh 1}}{\text{error of mesh 4}} \right) \Bigg/ \log_2\left(\sqrt[d]{\frac{\text{nodes in mesh 1}}{\text{nodes in mesh 4}}} \right), \tag{39}$$

where d is the spacial dimension.

6.1. 2D Results

In this section, we present the results of our 2-dimensional experiments. We use two different series of meshes, each series with 4 meshes. The first series of meshes (referred to collectively as "mesh series 1") is generated by placing nodes on a regular grid and then using MATLAB's Delaunay triangularization function to create the elements. The meshes range in size from 64×64 to 512×512 nodes. On the most refined mesh, the minimum angle is 45 degrees and the maximum
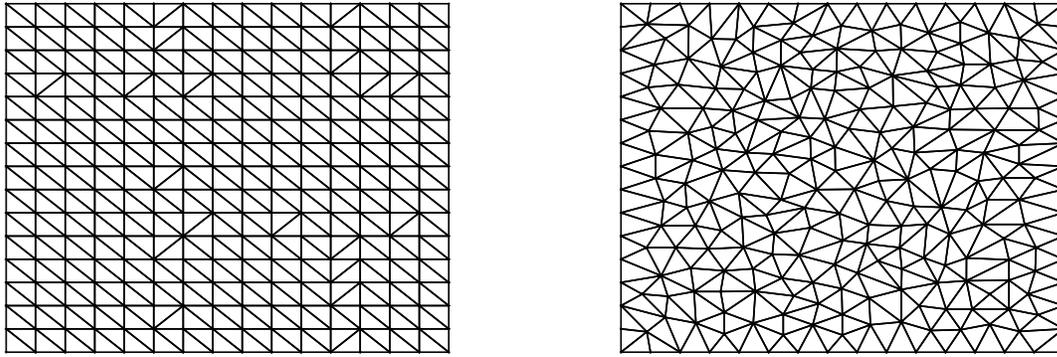
Figure 3. The mesh on the left is representative of the meshes used in series 1. The mesh on the right is representative of the meshes used in series 2. Note that the meshes above are coarser than the meshes used in computations so that the details can be seen clearly.

angle is 90 degrees. The maximum aspect ratio is 1.41, where a triangle's aspect ratio is defined as the ratio of the length of longest edge to the length of the smallest edge. The second series of meshes (referred collectively as "mesh series 2") is generated by using Triangle [45]. The number of nodes for each level of refinement is 4,103, 16,401, 65,655, and 262,597, respectively, approximately the same as those in series 1. On the most refined mesh, the maximum angle is 129.6 degrees and the minimum angle is 22.4 degrees. The maximum aspect ratio is 2.61. See Figure 3 for a visualization of the types of meshes used.

6.1.1. Poisson Equation The first set of results we present is for the Poisson equation with Dirichlet boundary conditions on the unit square. That is,

$$-\nabla^2 u = f \quad \text{in } \Omega = [0, 1]^2, \tag{40}$$

$$u = g \quad \text{on } \partial\Omega. \tag{41}$$

We consider the following three analytic solutions:

$$u_1 = 16x(1 - x)y(1 - y), \tag{42}$$

$$u_2 = \cos(\pi x)\cos(\pi y), \tag{43}$$

$$u_3 = \frac{1}{\sinh \pi \cosh \pi} \sinh(\pi x)\cosh(\pi y). \tag{44}$$

The Dirichlet boundary conditions are obtained from the given analytic solutions. The boundary conditions for u_1 are homogeneous and they are non-homogenous for u_2 and u_3 .

The L_∞ and L_2 norm errors for u_1 on mesh series 1 are displayed in Figure 4. One can see that the two graphs are fairly similar; this is true for u_2 and u_3 as well and thus we show only the L_∞ norm errors for these two problems; see Figure 5. GFD is the most accurate for u_1 and u_2 and AES-FEM 2 is the most accurate for u_3. FEM is the least accurate in all three cases.

For mesh series 2, the L_∞ and L_2 norm errors for u_1 can be seen in Figure 6 and the L_∞ norm errors for u_2 and u_3 can be seen in Figure 7. On this mesh series, AES-FEM 2 has the lowest error for u_1 and u_2. For u_3, the errors for GFD and AES-FEM 2 are very similar.
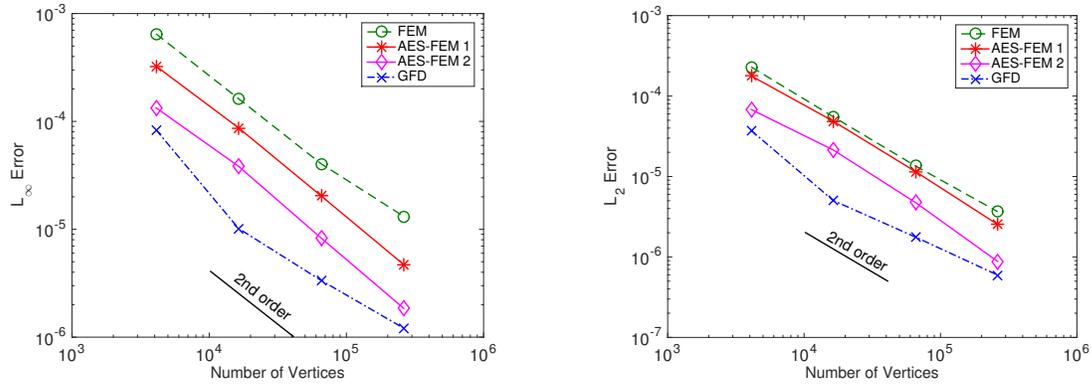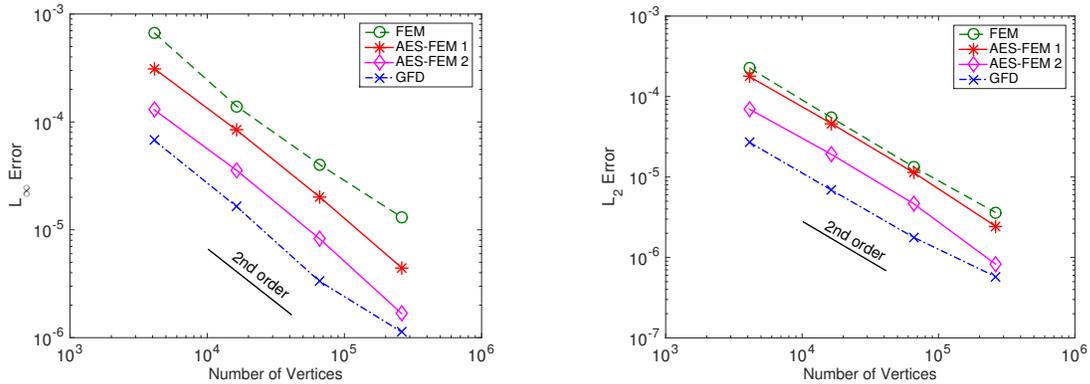
Figure 4. The errors for 2D Poisson equation on mesh 1 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).
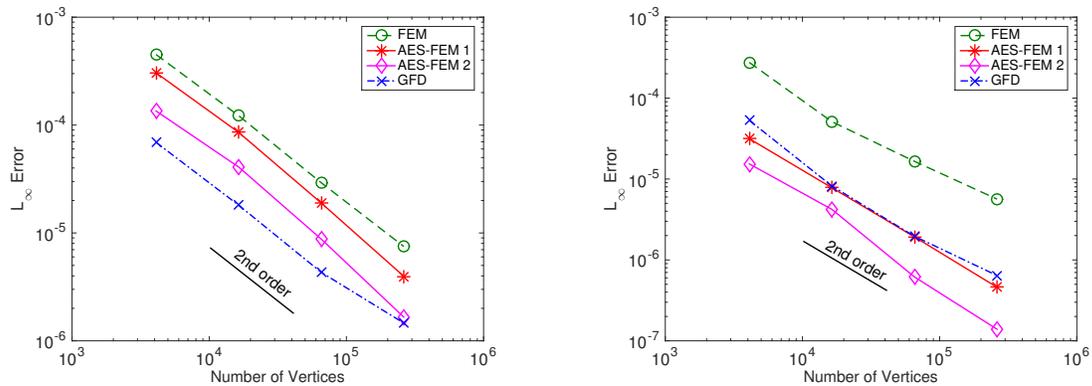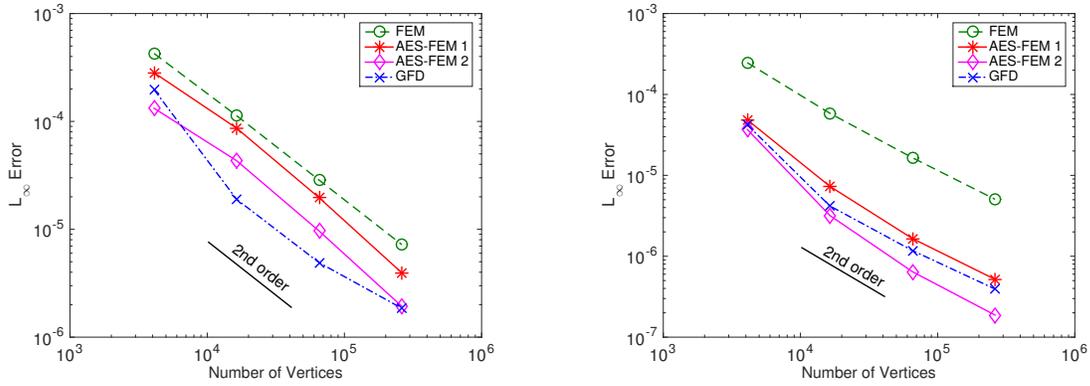

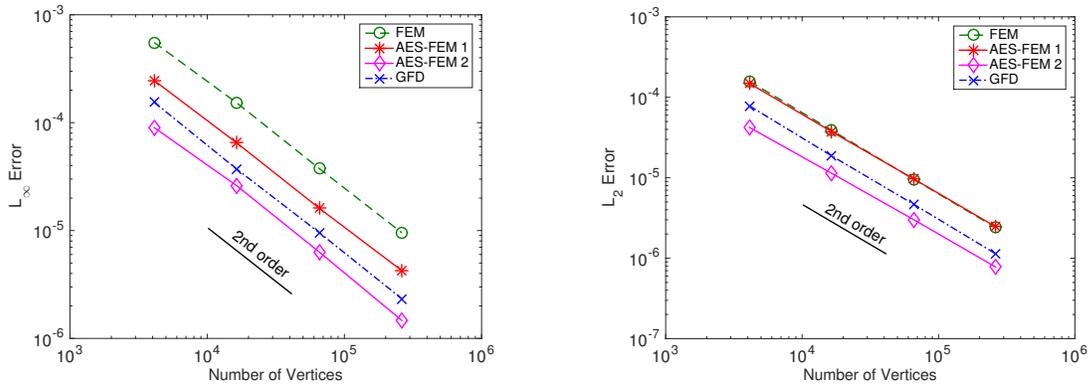Figure 5. The L_∞ norm errors for the 2D Poisson equation on mesh 1 for u_2 (left) and u_3 (right).
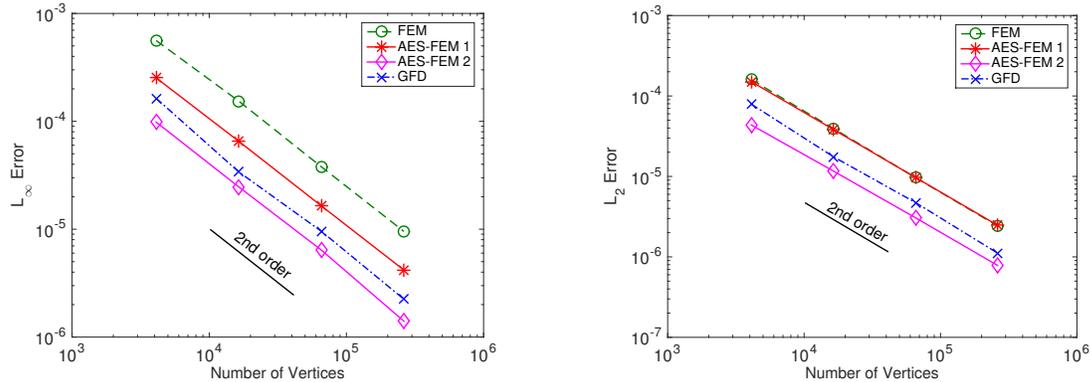
Figure 6. The errors for 2D Poisson equation on mesh 2 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).

6.1.2. Convection-Diffusion Equation We consider the convection-diffusion equation with Dirichlet boundary conditions on the unit square. That is,

$$-\nabla^2 u + c \cdot \nabla u = f \quad \text{in } \Omega, \tag{45}$$

$$u = g \quad \text{on } \partial\Omega. \tag{46}$$
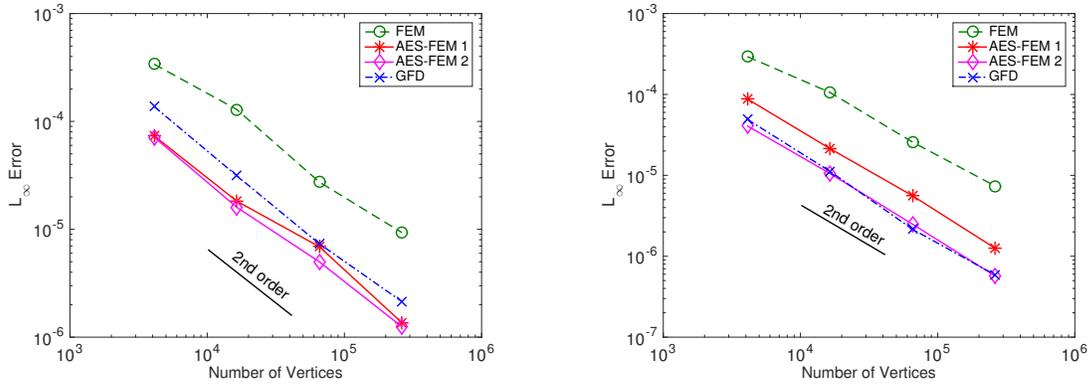
Figure 7. The L_∞ norm errors for the 2D Poisson equation on mesh 2 for u_2 (left) and u_3 (right).

Figure 8. The errors for 2D convection-diffusion equation on mesh 1 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).

We take $\boldsymbol{c} = [1, 1]^T$ for all of our tests and we consider the same analytic solutions as for the Poisson equation. Again the boundary conditions are obtained from the given analytic solutions.

The L_∞ and L_2 norm errors obtained from the convection-diffusion equation on mesh series 1 with u_1 are presented in Figure 8. The L_∞ norm errors for u_2 and u_3 on mesh series 1 are in Figure 9. For all three problems, AES-FEM and GFD are both more accurate than linear FEM. For u_1, GFD is the most accurate. For u_2, the most accurate method is either AES-FEM 2 or GFD depending on the level of refinement. For u_3, AES-FEM 2 is the most accurate.

On mesh series 2, AES-FEM 2 is the most accurate for u_1, as can be seen in Figure 10. For u_2 AES-FEM 1 or AES-FEM 2 is the most accurate, and for u_3 GFD is the most accurate; see Figure 11. In all these cases, AES-FEM is more accurate than FEM.

6.1.3. Element-Quality Dependence Test We test how FEM, AES-FEM, and GFD perform on a series of progressively worse meshes. We begin with the most refined mesh from mesh series 2. We select 6 of the 523,148 elements and incrementally move one of their nodes towards the opposite edge so as to create flatter triangles. We then solve the Poisson equation with the polynomial analytic solution u_1 in (42) and record the condition numbers of the coefficient and stiffness matrices and the numbers of iterations required for the solver to converge. Since the stiffness matrix is the same for

Figure 9. The L_∞ norm errors for the 2D convection-diffusion equation on mesh 1 for u_2 (left) and u_3 (right).

Figure 10. The errors for 2D convection-diffusion equation on mesh 2 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).
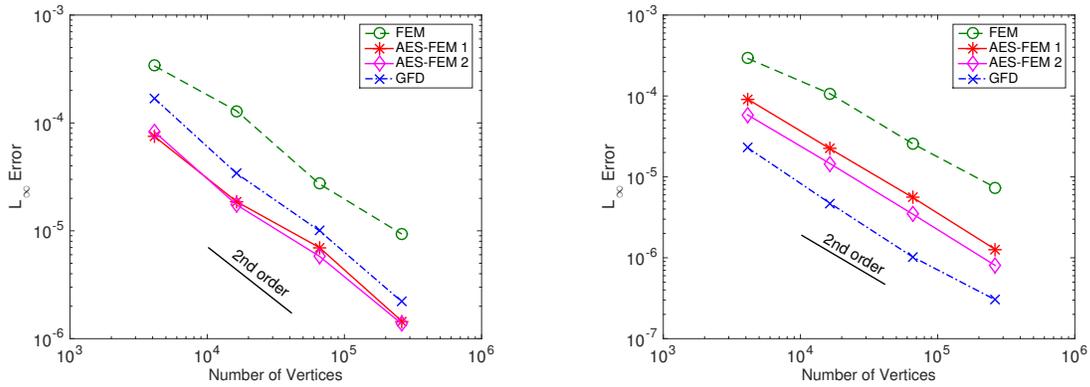

Figure 11. The L_∞ norm errors for the 2D convection-diffusion equation on mesh 2 for u_2 (left) and u_3 (right).

AES-FEM 1 and AES-FEM 2, the results are just labeled as AES-FEM. We use the conjugate gradient method with incomplete Cholesky preconditioner for FEM and we use GMRES with incomplete LU preconditioner for AES-FEM and GFD. The tolerance for the solvers is 10^{-8} and the drop tolerance for the preconditioners is 10^{-3}. As a measure of the mesh quality, we consider the cotangent of the minimum angle in the mesh; as the minimum angle tends to zero, the cotangent

Table I. Errors in L_2 norm for FEM, AES-FEM 1, AES-FEM 2, and GFD for u_1 on a series of meshes with progressively worse mesh element quality.

	FEM	AES-FEM 1	AES-FEM 2	GFD
Mesh 1	2.42×10^{-6}	2.47×10^{-6}	7.82×10^{-7}	1.11×10^{-6}
Mesh 2	2.42×10^{-6}	2.47×10^{-6}	7.83×10^{-7}	1.10×10^{-6}
Mesh 3	2.42×10^{-6}	2.47×10^{-6}	7.82×10^{-7}	1.10×10^{-6}
Mesh 4	2.42×10^{-6}	2.81×10^{-6}	1.19×10^{-6}	1.10×10^{-6}
Mesh 5	2.42×10^{-6}	2.81×10^{-6}	1.19×10^{-6}	1.10×10^{-6}
Mesh 6	2.42×10^{-6}	2.81×10^{-6}	1.19×10^{-6}	1.10×10^{-6}
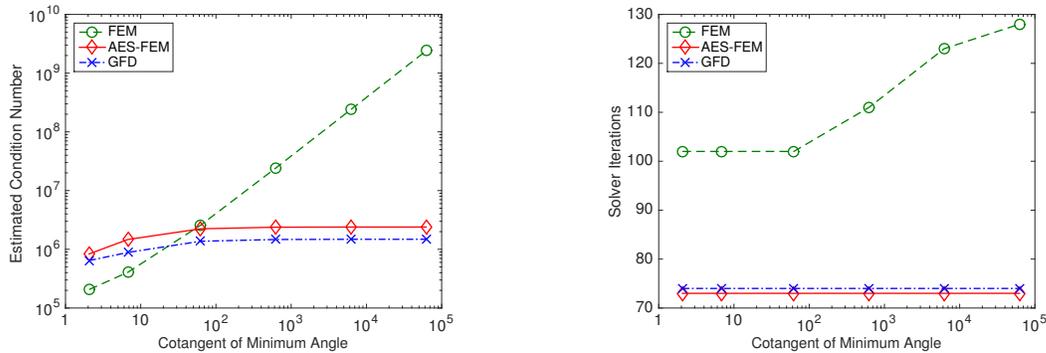
Figure 12. The condition numbers of the stiffness matrices for FEM and AES-FEM and the coefficient matrix for GFD (left) and the numbers of solver iterations (right). Solvers used are preconditioned CG for FEM and preconditioned GMRES for AES-FEM and GFD.

tends towards infinity. For very small angles, the cotangent of the angle is approximately equal to the reciprocal of the angle. We estimate the condition numbers using the MATLAB function `condest`, which computes a lower bound for the 1-norm condition number.

The worse the mesh quality, the higher the condition number of the stiffness matrix resulting from FEM. In contrast, the condition numbers of the GFD coefficient matrix and stiffness matrices from AES-FEM remain almost constant. As the condition number for FEM rises, so does the number of iterations required for the solver to converge, from 102 to 128. The numbers of iterations required to solve the equation for AES-FEM and GFD remain constant, at 73 and 74, respectively. We show the results for 6 meshes. Preconditioned conjugate gradient stagnates when trying to solve the FEM linear system from the 7th mesh, where the minimum angle is approximately 9.1×10^{-5} degrees. Solving the AES-FEM and GFD linear systems from the 7th mesh requires the same numbers of iterations as the other meshes, 73 and 74 respectively. See Figure 12 for a comparison of the condition numbers and the numbers of iterations.

The errors for both AES-FEM 1 and AES-FEM 2 rose slightly between the 3rd and the 4th mesh; the errors were then constant for the rest of the meshes. The errors for FEM remained constant and the errors for GFD remained nearly constant over the 6 meshes. AES-FEM 1, AES-FEM 2 and GFD converge on the 7th mesh in the series with the same errors as on Mesh 6, whereas for FEM the solver stagnates. See Table I for specific errors.

6.1.4. Efficiency We compare the runtimes for the four methods: AES-FEM 1, AES-FEM 2, FEM, and GFD. We consider the convection-diffusion equation on the most refined mesh of series 2
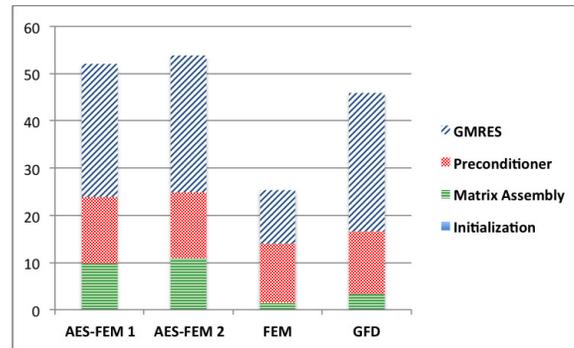
Figure 13. Runtimes for a 2D convection-diffusion equation on the most refined mesh in series 2.

with the polynomial analytic solution u_1 for this runtime experiment. We decompose the total time into 4 subcategories: *Initialization,* which includes the time to load the mesh and the time to assign the boundary conditions and problem values; *Assembly,* which includes the time to build the stiffness matrix and load vector; *Preconditioner,* which is the time it takes to construction the matrix preconditioner using incomplete LU factorization with a drop tolerance of 10^{-3}; and *Solver,* which is the amount of time for solving the preconditioned system using GMRES with a tolerance of 10^{-8}. See Figure 13 for the comparison. The initialization time is minuscule compared to the other categories and is not visible in the figure. FEM requires 76 iterations of GMRES to converge, AES-FEM 1 and AES-FEM 2 both require 74 iterations, and GFD requires 75 iterations.

One can see that FEM has an advantage when it comes to efficiency on the same mesh. In terms of total runtime, it is about 2.1 times faster than AES-FEM 1, 2.1 times faster than AES-FEM 2, and 1.8 times faster than GFD. In terms of assembly time, FEM is about 6.1 times faster than AES-FEM 1, 7.7 times faster than AES-FEM 2, and 2.2 times faster than GFD. Comparing the assembly time of AES-FEM and GFD, we see that GFD is approximately 3.2 and 3.5 times faster than AES-FEM 1 and AES-FEM 2, respectively.

In 2D, assembling the load vector using FEM basis functions (AES-FEM 1) saves some time compared to using GLP basis functions (AES-FEM 2). AES-FEM 1 offers a savings of approximately 1.1 seconds or, in other words, a 10.3% reduction of assembly time and a 3.5% reduction of total time compared to AES-FEM 2. We will see in the next section that the efficiency of these two methods varies more in 3D.

However, in terms of error versus runtime, AES-FEM is competitive with, and often is more efficient than, the classical FEM with linear basis functions. In Figure 14, we compare the L_∞ norm errors versus runtimes for the four methods on mesh series 2 for the Poisson equation and the convection-diffusion equation with the analytic solution equal to u_2. For the Poisson equation, all four methods are very similar, with AES-FEM 1 being slightly more efficient for finer meshes. For the convection-diffusion equation, AES-FEM 1 and AES-FEM 2 are approximately the same in terms of efficiency and are the most efficient. GFD is also more efficient than FEM.

6.2. 3D Results

In this section, we present the results from the 3D experiments. We consider the Poisson equation and the convection-diffusion equation in 3D. We test three problems for each equation on two
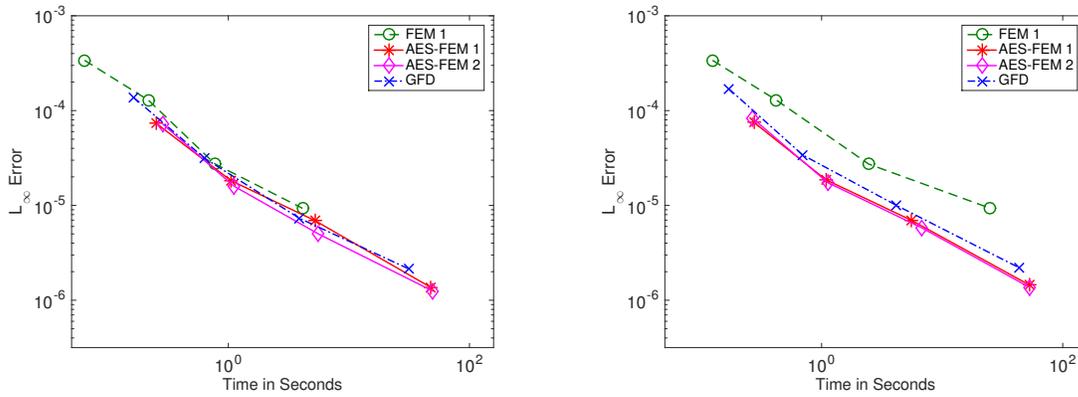
Figure 14. L_∞ norm errors versus runtimes for a 2D Poisson equation (left) and convection-diffusion equation (right) on mesh series 2. Lower is better.

different series of meshes, each with four levels of refinement. The first series of meshes (referred to collectively as "mesh series 1") is created by placing nodes on a regular grid and using MATLAB's Delaunay triangularization to create the elements. The meshes in series 1 range from $8 \times 8 \times 8$ nodes to $64 \times 64 \times 64$ nodes. The minimum dihedral angle in the most refined mesh of series 1 is 35.2 degrees and the maximum dihedral angle is 125.2 degrees. The maximum aspect ratio is 4.9, where the aspect ratio of a tetrahedron is defined as the ratio of the longest edge length to the smallest height. The second series of meshes (referred to collectively as "mesh series 2") is created using TetGen [3]. The number of nodes in mesh series 2 for each level of refinement is 509, 4,080, 32,660, and 261,393, which is approximately the same as the meshes in series 1. The minimum dihedral angle of the most refined mesh in series 2 is 6.7 degrees and the largest dihedral angle is 165.5 degrees. The largest aspect ratio is 15.2.

6.2.1. Poisson Equation We first consider the Poisson equation with Dirichlet boundary conditions on the unit cube. That is,

$$-\nabla^2 u = f \quad \text{in } \Omega, \tag{47}$$

$$u = g \quad \text{on } \partial\Omega. \tag{48}$$

where $\Omega = [0,1]^3$. We consider three different analytic solutions listed below.

$$u_1 = 64x\left(1-x\right)y\left(1-y\right)z\left(1-z\right), \tag{49}$$

$$u_2 = \cos(\pi x)\cos(\pi y)\cos(\pi z), \tag{50}$$

$$u_3 = \frac{1}{\sinh\pi\cosh\pi\cosh\pi}\sinh(\pi x)\cosh(\pi y)\cosh(\pi z). \tag{51}$$

The Dirichlet boundary conditions are derived from the analytic solutions. They are homogeneous for u_1 and non-homogeneous for u_2 and u_3.

The L_∞ and L_2 norm errors for the Poisson equation on mesh series 1 for u_1 can be seen in Figure 15 and the L_∞ norm errors for u_2 and u_3 can be seen in Figure 16. GFD is the most accurate for u_1 and u_2. AES-FEM 2 is the most accurate for u_3.
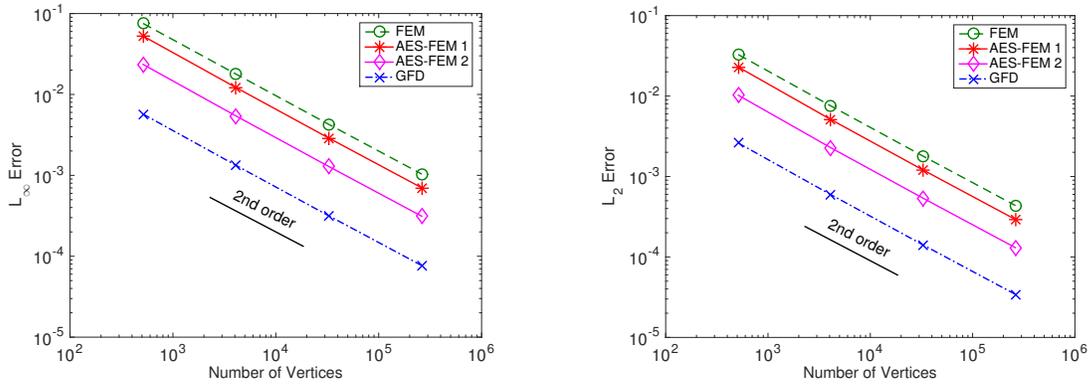
Figure 15. The errors for 3D Poisson equation on mesh 1 with u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).
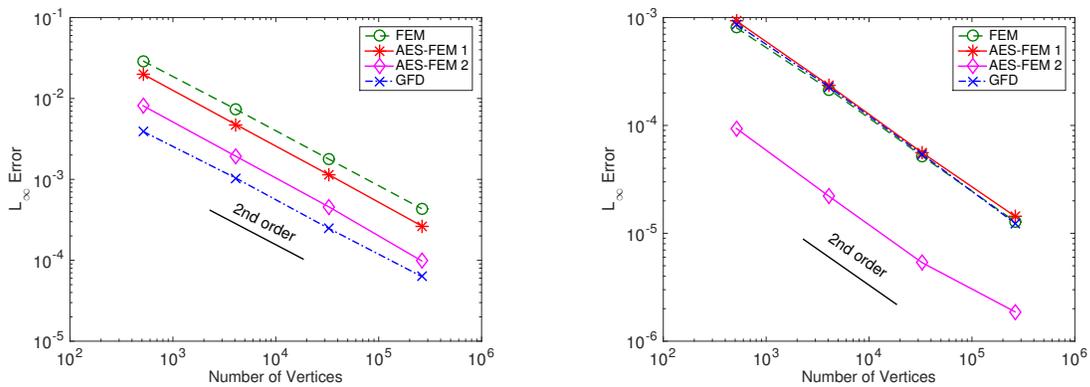

Figure 16. The L_∞ norm errors for the 3D Poisson equation on mesh 1 for u_2 (left) and u_3 (right).
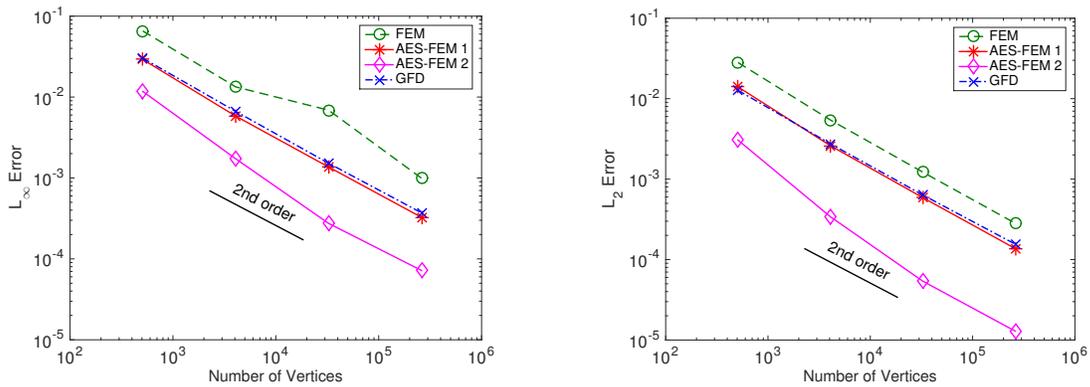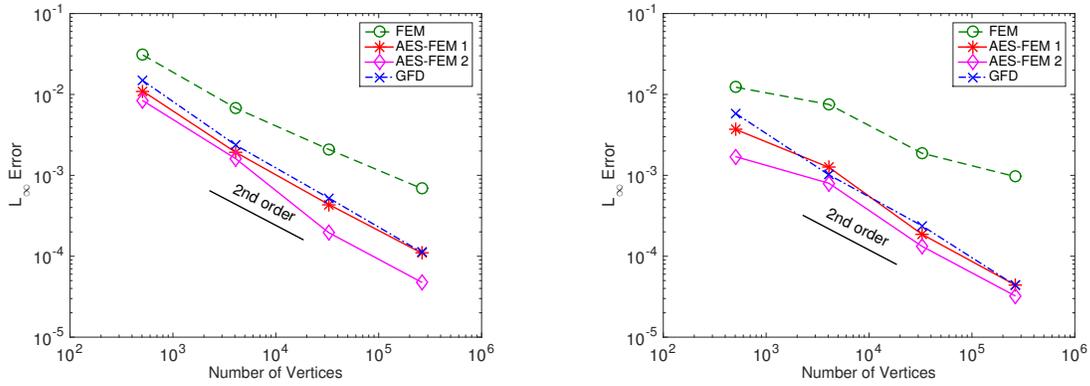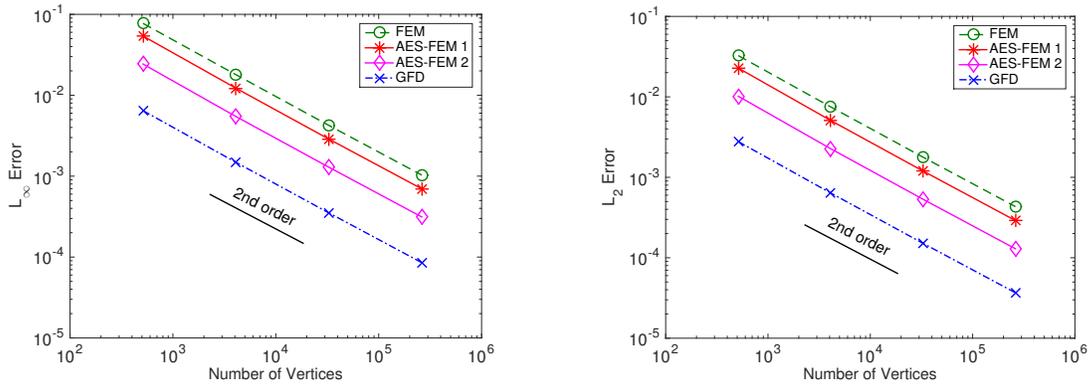

Figure 17. The errors for 3D Poisson equation on mesh 2 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).

For mesh series 2, AES-FEM 2 is the most accurate and is close to an order of magnitude more accurate than FEM. See Figure 17 for the L_∞ and L_2 norm errors for u_1 and Figure 18 for the L_∞ norm errors for u_2 and u_3.

Figure 18. The L_∞ norm errors for the 3D Poisson equation on mesh 2 for u_2 (left) and u_3 (right).

Figure 19. The errors for 3D convection-diffusion equation on mesh 1 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).

6.2.2. Convection-Diffusion Equation We consider the convection-diffusion equation with Dirichlet boundary conditions on the unit cube, $\Omega = [0, 1]^3$.

$$-\nabla^2 u + c \cdot \nabla u = f \quad \text{in } \Omega, \tag{52}$$

$$u = g \quad \text{on } \partial\Omega. \tag{53}$$

We take $c = [1, 1, 1]^T$ and we consider the same analytic solutions as in the previous section. Again, the Dirichlet boundary conditions are derived from the analytic solutions u_1, u_2, and u_3.

The L_∞ and L_2 norm errors for the 3D convection-diffusion equation on mesh series 1 for u_1 can be seen in Figure 19, see Figure 20 for the L_∞ norm errors on mesh series 1 for u_2 and u_3. As with the Poisson equation, GFD is the most accurate for u_1 and u_2. For u_3, either GFD or AES-FEM 2 is the most accurate depending on the level of refinement.

On mesh series 2, AES-FEM 2 is the most accurate for u_1 and u_2, as can be seen in Figure 21 and the left panel of Figure 22. For u_3, either GFD or AES-FEM 2 is the most accurate, as can be seen in the right panel of Figure 22. Similar to 2D results, AES-FEM is more accurate than FEM in all these cases.
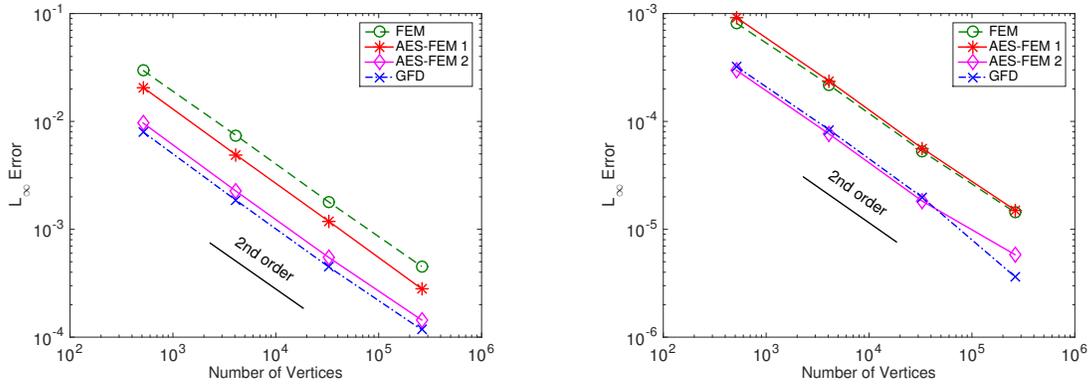
Figure 20. The L_∞ norm errors for the 3D convection-diffusion equation on mesh 1 for u_2 (left) and u_3 (right).
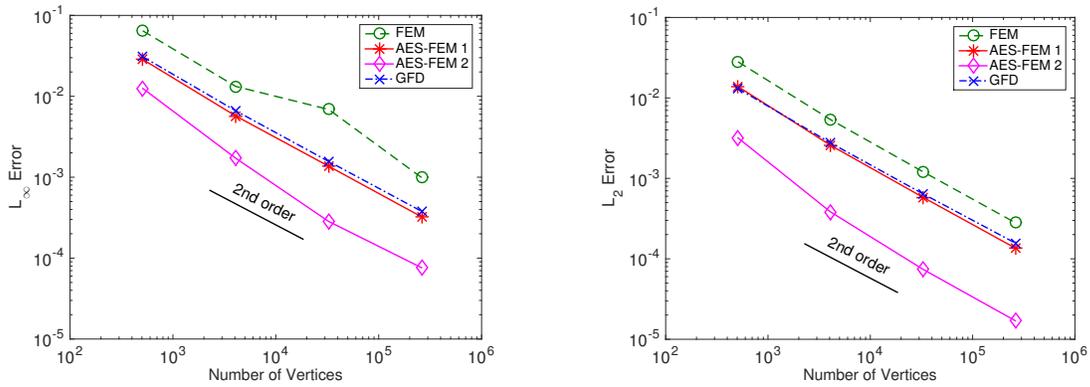

Figure 21. The errors for 3D convection-diffusion equation on mesh 2 for u_1. The errors were computed using the L_∞ norm (left) and the L_2 norm (right).
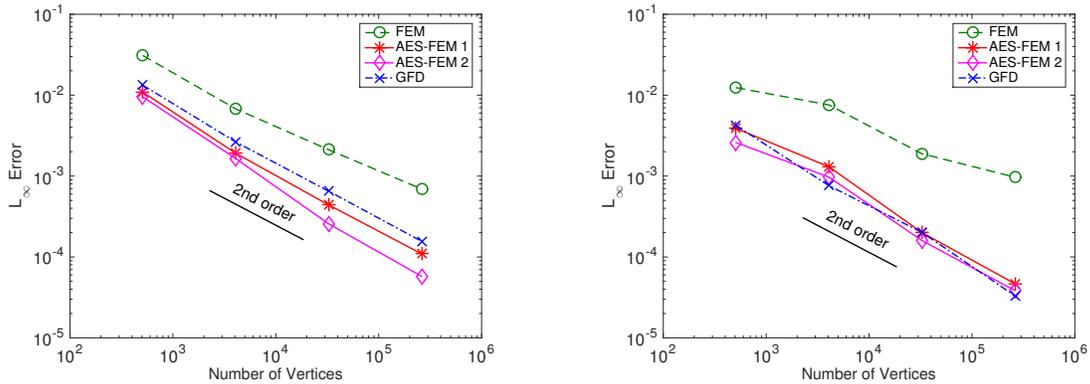

Figure 22. The L_∞ errors for the 3D convection-diffusion equation on mesh 2 for u_2 (left) and u_3 (right).

6.2.3. Element-Quality Dependence Test   We test how FEM, AES-FEM, and GFD perform on a series of meshes with progressively worse element shape quality. We begin with the most refined mesh from mesh series 1. We select 69 out of the 1,500,282 elements and incrementally move one of their nodes towards the opposite side so as to create sliver tetrahedra. We then solve the Poisson equation with the polynomial analytic solution u_1 in (49) and record the condition numbers of the coefficient and stiffness matrices and the numbers of iterations required for the solver to
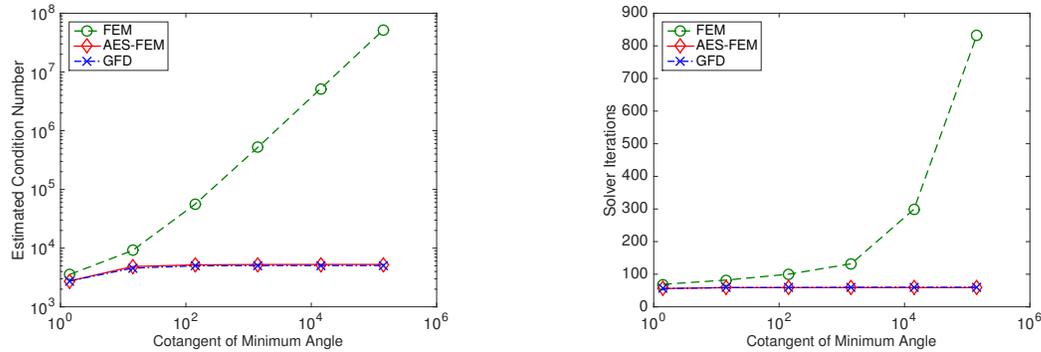
Figure 23. The condition numbers of the stiffness matrices for FEM and AES-FEM and the coefficient matrix for GFD (left) and the numbers of solver iterations (right). Solvers used are preconditioned CG for FEM and preconditioned GMRES for AES-FEM and GFD.

converge. We use the conjugate gradient method with Gauss-Seidel preconditioner for FEM and we use GMRES with Gauss-Seidel preconditioner for AES-FEM and GFD. We use a tolerance of 10^{-5} for both solvers. As a measure of the mesh quality, we consider the cotangent of the minimum dihedral angle in the mesh; as the minimum angle tends to zero, the cotangent tends towards infinity. For very small angles, the cotangent of the angle is approximately equal to the reciprocal of the angle. We estimate the condition numbers using the MATLAB function `condest`, which computes a lower bound for the 1-norm condition number.

The worse the mesh quality, the higher the condition number of the stiffness matrix resulting from FEM. The condition numbers of the stiffness matrices from AES-FEM and the coefficient matrix from GFD remain almost constant. As the condition number of the matrix rises so does the number of iterations required for the solver to converge. For FEM the number of iterations increases from 69 for the best mesh to 831 for the most deformed mesh. The numbers of iterations for AES-FEM and GFD remain almost constant, increasing from 56 to 59 and from 56 to 60, respectively. See Figure 23 for a comparison of the condition numbers and iteration counts of the solvers.

For each of the four methods, the errors were nearly constant over the series of meshes. For FEM, the L_2 error on the 1st mesh was 4.36×10^{-4} and on the 6th mesh, the error was 4.37×10^{-4}. For AES-FEM 1, the L_2 error on all the meshes was 2.92×10^{-4}. For AES-FEM 2, the L_2 error on all the meshes was 1.30×10^{-4}. For GFD, the L_2 error on the 1st mesh was 3.43×10^{-5} and on the 6th mesh, the error was 3.40×10^{-5}.

6.2.4. Efficiency We compare the runtimes of the four methods for solving the convection-diffusion equation with the polynomial analytic solution u_1 on the most refined mesh of series 2. As in 2D, the total time is decomposed into 4 subcategories: *Initialization, Assembly*, *Preconditioner*, and *Solver*. The preconditioner used is incomplete LU with a drop tolerance of 10^{-1}. GMRES with a tolerance of 10^{-8} is used as the solver. AES-FEM 1 and AES-FEM 2 each require 142 iterations to converge, FEM requires 128 iterations, and GFD requires 138 iterations. The majority of the time is spent assembling the matrix and solving the system. See Figure 24 for the comparison.

As in 2D, FEM is the most efficient method on a given mesh. Overall, the total runtime of FEM is approximately 1.7 times faster than AES-FEM 1, 1.9 times faster than AES-FEM 2, and 1.8 times
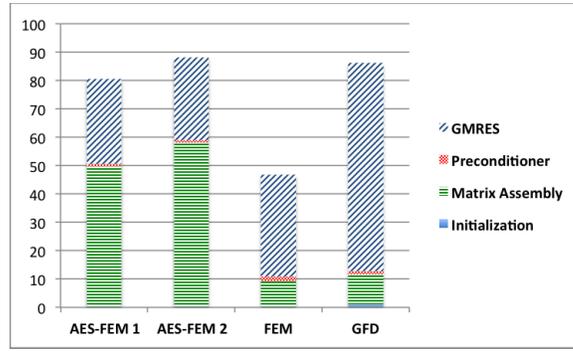
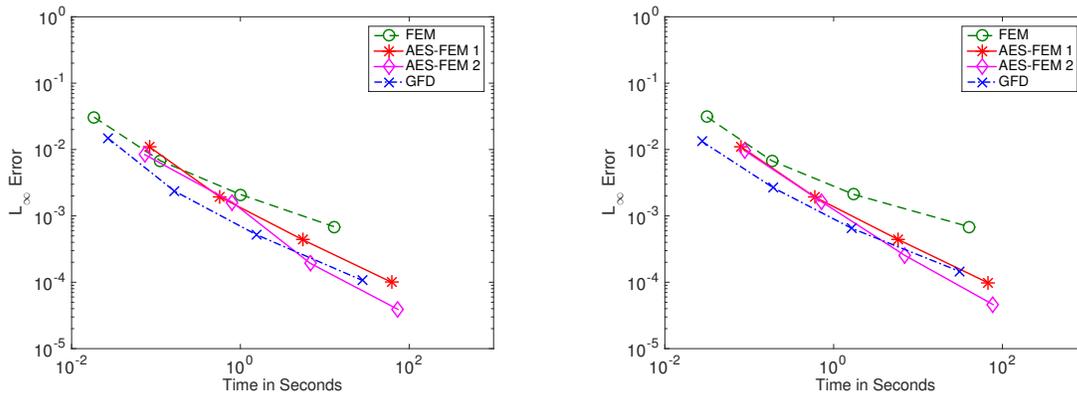Figure 24. Runtimes for a 3D convection-diffusion equation on the most refined mesh in series 2.

Figure 25. L_∞ norm errors versus runtimes for a 3D Poisson equation (left) and convection-diffusion equation (right) on mesh series 2. Lower is better.

faster than GFD. The assembly of FEM is approximately 5.6 times faster than AES-FEM 1, 6.5 times faster than AES-FEM 2, and 1.2 times faster than GFD.

In 3D, the difference of assembling the load vector using FEM basis functions (AES-FEM 1) versus using GLP basis functions (AES-FEM 2) is more pronounced than in 2D. The assembly in AES-FEM 1 is 8.3 seconds shorter than that of AES-FEM 2. This means the assembly of AES-FEM 1 uses 14.4% less time than that of AES-FEM 1 and the total runtime is 8.7% shorter.

However, similar to 2D, AES-FEM is competitive with, and most of time more efficient than, the classical FEM with linear basis functions in terms of error versus runtime. Figure 25 shows the L_∞ norm errors versus runtimes for the four methods on mesh series 2 for the Poisson equation and the convection-diffusion equation for u_2. For the Poisson equation, GFD is more efficient on coarser meshes and AES-FEM 2 is more efficient for finer meshes. For the convection-diffusion equation, GFD is more efficient on smaller meshes and AES-FEM 2 is more efficient for finer meshes. AES-FEM 1 is also more efficient than FEM.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an adaptive extended stencil finite element method, which uses generalized Lagrange polynomial basis functions constructed from weighted least squares

approximations. The method preserves the theoretical framework of the classical FEM and the simplicity in imposing essential boundary conditions and integrating the stiffness matrix. We presented the formulation of AES-FEM, showed that the method is consistent, and discussed both the local and global stability of the method. We described the implementation, including the mesh data structure and the numerical algorithms. We compared the accuracy of AES-FEM against the classical FEM with linear basis functions and the quadratic generalized finite difference methods for the Poisson and convection-diffusion equations in both 2D and 3D. We showed improved accuracy and stability of AES-FEM over FEM, and demonstrated that the condition number of AES-FEM, and hence the convergence rate of iterative solvers, are independent of the element quality of the mesh. Our experiments also showed that AES-FEM is more efficient than the classical FEM in terms of error versus runtime, while having virtually the same sparsity patterns of the stiffness matrices.

As a general method, AES-FEM can use generalized Lagrange polynomial basis functions of arbitrary degrees. We only focused on quadratic basis functions in this paper. In future work, we will report higher-order AES-FEM with cubic and higher-degree basis functions. The present implementation of AES-FEM uses the standard hat functions as the weight functions, which may lead to large errors when applied to tangled meshes with inverted elements. We will report the resolution of tangled meshes in a future publication. Finally, while AES-FEM is efficient in terms of error versus runtime, it is much slower than the classical FEM on a given mesh due to the slower computation of the basis functions and the nonsymmetry of the stiffness matrix. The efficiency can be improved substantially by leveraging the parallelism and the efficient multigrid solvers, which we will report in the future.

References

1. Zienkiewicz O, Taylor R, Zhu J. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 2013.
2. Shewchuk JR. Delaunay refinement algorithms for triangular mesh generation. *Comput. Geom.* 2002; **22**(1):21–74, doi:10.1016/S0925-7721(01)00047-5.
3. Si H. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Software* 2015; **41**(2):11:1 – 11:36, doi:10.1145/2629697.
4. Lo S. A new mesh generation scheme for arbitrary planar domains. *Int. J. Numer. Meth. Engrg.* 1985; **21**(8):1403– 1426, doi:10.1002/nme.1620210805.
5. Shephard MS, Georges MK. Automatic three-dimensional mesh generation by the finite octree technique. *Int. J. Numer. Meth. Engrg.* 1991; **32**(4):709–749, doi:10.1002/nme.1620320406.
6. Patera AT. A spectral element method for fluid dynamics: Laminar flow in a channel expansion. *J. Comput. Phys.* 1984; **54**(3):468–488, doi:10.1016/0021-9991(84)90128-1.
7. Arnold DN, Brezzi F, Cockburn B, Marini LD. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.* 2002; **39**(5):1749–1779, doi:10.1137/S0036142901384162.

8. Cockburn B, Karniadakis GE, Shu CW. *The Development of Discontinuous Galerkin Methods*. Springer, 2000.

9. Belytschko T, Lu YY, Gu L. Element free Galerkin methods. *Int. J. Numer. Meth. Engrg.* 1994; **37**(2):229–256, doi:10.1002/nme.1620370205.

10. Nayroles B, Touzot G, Villon P. Generalizing the finite element method: Diffuse approximation and diffuse elements. *Comput. Mech.* 1992; **10**(5):307–318, doi:10.1007/BF00364252.

11. Bochev PB, Gunzburger MD. Finite element methods of least-squares type. *SIAM Rev.* 1998; **40**(4):789–837, doi:10.1137/S0036144597321156.

12. Jensen PS. Finite difference techniques for variable grids. *Comput. Struct.* 1972; **2**(1):17–29, doi:10.1016/0045-7949(72)90020-X.

13. Perrone N, Kao R. A general finite difference method for arbitrary meshes. *Comput. Struct.* 1975; **5**(1):45–57, doi:10.1016/0045-7949(75)90018-8.

14. Liszka T, Orkisz J. The finite difference method at arbitrary irregular grids and its application in applied mechanics. *Comput. Struct.* 1980; **11**(1):83–95, doi:10.1016/0045-7949(80)90149-2.

15. Benito J, Ureña F, Gavete L. Influence of several factors in the generalized finite difference method. *Appl. Math. Model.* 2001; **25**:1039–1053, doi:10.1016/S0307-904X(01)00029-4.

16. Milewski S. Meshless finite difference method with higher order approximation applications in mechanics. *Arch. Comput. Method. E.* 2012; **19**:1–49, doi:10.1007/s11831-012-9068-y.

17. Belytschko T, Gracie R, Ventura G. A review of extended/generalized finite element methods for material modeling. *Model. Simul. Mater. Sci. Eng.* 2009; **17**(4):043 001, doi:10.1088/0965-0393/17/4/043001/.

18. Fries TP, Belytschko T. The extended/generalized finite element method: An overview of the method and its applications. *Int. J. Numer. Meth. Engrg.* 2010; **84**(3):253–304, doi:10.1002/nme.2914.

19. Melenk JM, Babuška I. The partition of unity finite element method: Basic theory and applications. *Comput. Method. Appl. M.* 1996; **139**(1):289–314, doi:10.1016/S0045-7825(96)01087-0.

20. Lancaster P, Salkauskas K. Surfaces generated by moving least squares methods. *Math. Comp.* 1981; **37**(155):141–158, doi:10.2307/2007507.

21. Duarte CA, Oden JT. An h-p adaptive method using clouds. *Comput. Meth. Appl. Mech. Engrg.* 1996; **139**(1):237–262, doi:10.1016/S0045-7825(96)01085-7.

22. Duarte C, Babuska I, Oden J. Generalized finite element methods for three-dimensional structural mechanics problems. *Comput. Struct.* 2000; **77**(2):215 – 232, doi:10.1016/S0045-7949(99)00211-4.

23. Srinivasan K, Matouš K, Geubelle P. Generalized finite element method for modeling nearly incompressible bimaterial hyperelastic solids. *Comput. Meth. Appl. Mech. Engrg.* 2008; **197**(51):4882–4893, doi:10.1016/j.cma.2008.07.014.

24. Hughes TJ, Cottrell JA, Bazilevs Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Meth. Appl. Mech. Engrg.* 2005; **194**(39):4135–4195, doi:10.1016/j.cma.2004.10.008.

25. Forsythe GE, Wasow WR. *Finite-Difference Methods for Partial Differential Equations*. John Wiley & Sons, Inc: New York - London, 1960.

26. Demkowicz L, Karafiat A, Liszka T. On some convergence results for FDM with irregular mesh. *Comp. Meth. Appl. Mech. Engrg* 1984; **42**(3):343–355, doi:10.1016/0045-7825(84)90013-6.

27. Gavete L, Gavete M, Benito J. Improvements of generalized finite difference method and comparison with other meshless methods. *Appl. Math. Model.* 2003; **27**:831–847, doi:10.1016/S0307-904X(03)00091-X.

28. Benito J, Ureña F, Gavete L. Solving parabolic and hyperbolic equations by the generalized finite difference method. *J. Comput. Appl. Math.* 2007; **209**(2):208–233, doi:10.1016/j.cam.2006.10.090.

29. Prieto FU, Benito Muñoz JJ, Corvinos LG. Application of the generalized finite difference method to solve the advection–diffusion equation. *J. Comput. Appl. Math.* 2011; **235**(7):1849–1855, doi:doi:10.1016/j.cam.2010.05.026.

30. Milewski S. Selected computational aspects of the meshless finite difference method. *Numer. Algorithms* 2013; **63**(1):107–126, doi:10.1007/s11075-012-9614-6.

31. Jiao X, Zha H. Consistent computation of first-and second-order differential quantities for surface meshes. *ACM Symposium on Solid and Physical Modeling*, ACM, 2008; 159–170, doi:10.1145/1364901.1364924.

32. Wang D, Clark B, Jiao X. An analysis and comparison of parameterization-based computation of differential quantities for discrete surfaces. *Comput. Aid. Geom. Des.* 2009; **26**(5):510–527, doi:10.1016/j.cagd.2009.02.006.

33. Brenner SC, Scott R. *The Mathematical Theory of Finite Element Methods*, vol. 15. Springer Science & Business Media, 2008.

34. Ciarlet PG. *The Finite Element Method for Elliptic Problems*. 2nd edn., SIAM, 2002.

35. Finlayson BA. *The Method of Weighted Residuals and Variational Principles*. Academic Press: New York, 1973.

36. Golub GH, Van Loan CF. *Matrix Computations*. 4th edn., Johns Hopkins, 2013.

37. Cazals F, Pouget M. Estimating differential quantities using polynomial fitting of osculating jets. *Comput. Aid. Geom. Des.* 2005; **22**(2):121–146, doi:10.1016/j.cagd.2004.09.004.

38. Babuška I, Aziz AK. On the angle condition in the finite element method. *SIAM J. Numer. Anal.* 1976; **13**(2):214–226, doi:10.1137/0713021.

39. Lui SH. *Numerical Analysis of Partial Differential Equations*. John Wiley & Sons, 2012.

40. Watkins DS. *Fundamentals of Matrix Computations*. John Wiley & Sons, 2004.

41. Dyedov V, Ray N, Einstein D, Jiao X, Tautges TJ. AHF: Array-based half-facet data structure for mixed-dimensional and non-manifold meshes. *Proceedings of the 22nd International Meshing Roundtable*. Springer, 2014; 445–464, doi:10.1007/978-3-319-02335-9_25.

42. Higham NJ. A survey of condition number estimation for triangular matrices. *SIAM Rev.* 1987; **29**(4):575–596, doi:10.1137/1029112.

43. Saad Y, Schultz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.* 1986; **7**(3):856–869, doi:10.1137/0907058.

44. Trottenberg U, Oosterlee CW, Schuller A. *Multigrid*. Academic press, 2000.

45. Shewchuk JR. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. *Applied Computational Geometry Towards Geometric Engineering*. Springer, 1996; 203–222, doi:10.1007/BFb0014497.