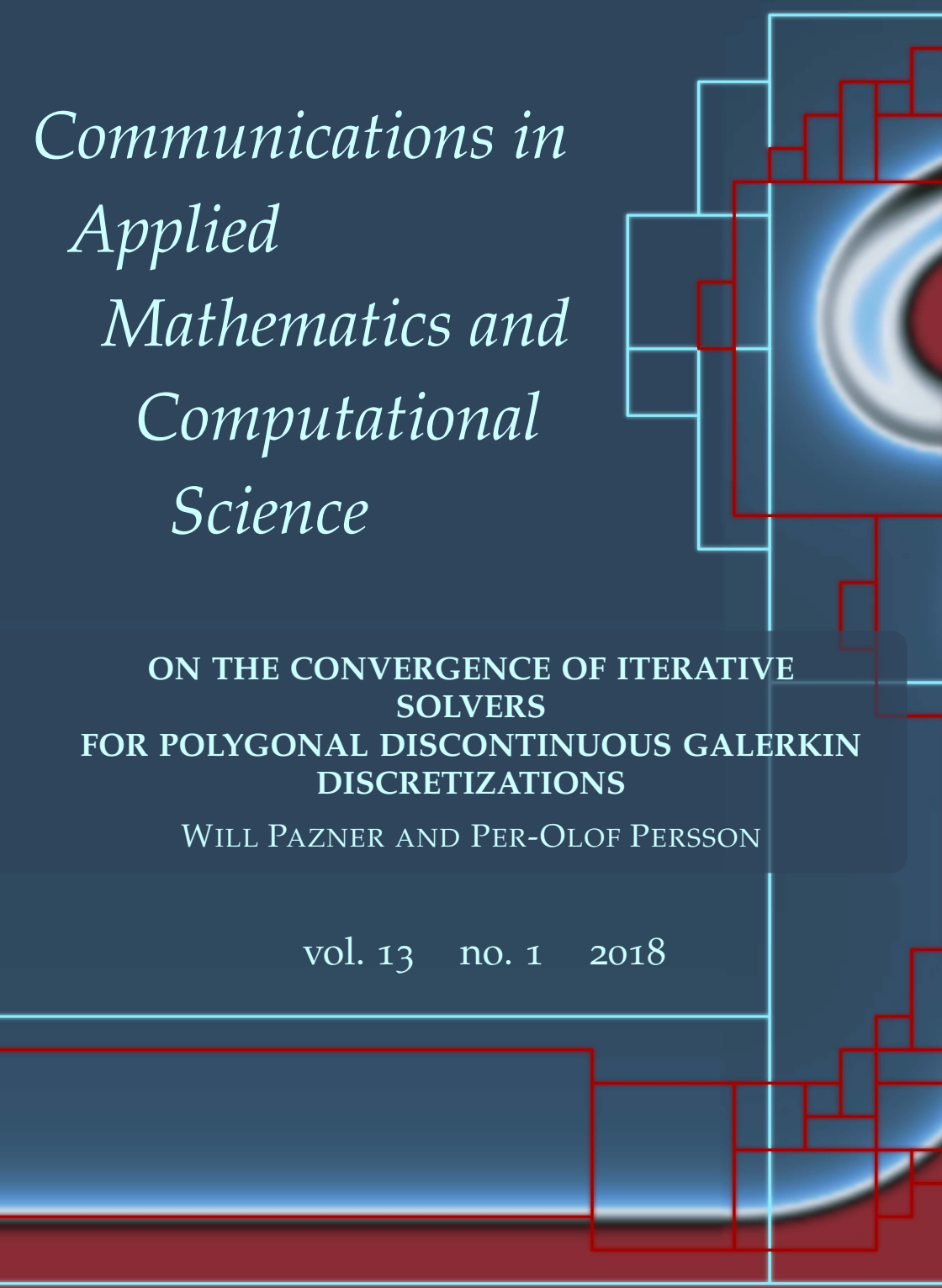
Communications in Applied Mathematics and Computational Science

ON THE CONVERGENCE OF ITERATIVE SOLVERS FOR POLYGONAL DISCONTINUOUS GALERKIN DISCRETIZATIONS

WILL PAZNER AND PER-OLOF PERSSON

vol. 13 no. 1 2018



msp

COMM. APP. MATH. AND COMP. SCI.
Vol. 13, No. 1, 2018
dx.doi.org/10.2140/camcos.2018.13.27

msp

ON THE CONVERGENCE OF ITERATIVE SOLVERS FOR POLYGONAL DISCONTINUOUS GALERKIN DISCRETIZATIONS

WILL PAZNER AND PER-OLOF PERSSON

We study the convergence of iterative linear solvers for discontinuous Galerkin discretizations of systems of hyperbolic conservation laws with polygonal mesh elements compared with traditional triangular elements. We solve the semidiscrete system of equations by means of an implicit time discretization method, using iterative solvers such as the block Jacobi method and GMRES. We perform a von Neumann analysis to analytically study the convergence of the block Jacobi method for the two-dimensional advection equation on four classes of regular meshes: hexagonal, square, equilateral-triangular, and right-triangular. We find that hexagonal and square meshes give rise to smaller eigenvalues, and thus result in faster convergence of Jacobi's method. We perform numerical experiments with variable velocity fields, irregular, unstructured meshes, and the Euler equations of gas dynamics to confirm and extend these results. We additionally study the effect of polygonal meshes on the performance of block ILU(0) and Jacobi preconditioners for the GMRES method.

1. Introduction

In recent years, the discontinuous Galerkin (DG) method has become a popular choice for the discretization of a wide range of partial differential equations [27; 6; 15]. This is partly because of its many attractive properties, such as the arbitrarily high degrees of approximation, the rigorous theoretical foundation, and the ability to use fully unstructured meshes. Also, due to its natural stabilization mechanism based on approximate Riemann solvers, it has in particular become widely used in fluid dynamics applications where the high-order accuracy is believed to produce improved accuracy for many problems [32].

Most work on DG methods has been based on meshes of either simplex elements (triangles and tetrahedra), block elements (quadrilaterals and hexahedra), or combinations of these such as prism elements. This is likely because of the availability of excellent automatic unstructured mesh generators, at least for the simplex case

MSC2010: 65F10, 65M60, 65N22.
Keywords: discontinuous Galerkin, iterative solvers, preconditioners.

[22; 28; 30], and also because of the advantages with the outer product structure of block elements. However, it is well known that since no continuity is enforced between the elements, it is straightforward to apply the DG methods to meshes with elements of any shapes (even nonconforming ones). For example, vertex-centered DG methods based on the polygonal dual meshes were studied in [5; 18]. This is a major advantage over standard continuous FEM methods, which need significant developments for the extension to arbitrary polygonal and polyhedral elements [19].

In the finite volume CFD community, there has recently been considerable interest in meshes of arbitrary polygonal and polyhedral elements. In fact, the popular vertex-centered finite volume method applied to a tetrahedral mesh can be seen as a cell-centered method on the dual polyhedral mesh. Because of this, a number of methods have been proposed for generation of polyhedral meshes, which in many cases have advantages over traditional simplex meshes [21; 12]. Although it is still unclear exactly what benefits these elements provide, they have been reported to be both more accurate per degree of freedom and to have better convergence properties in the numerical solvers than for a corresponding tetrahedral mesh [23; 2]. There have also been studies showing that vertex-centered schemes are preferred over cell-centered [10; 9], again indicating the benefits of polyhedral elements.

Inspired by the promising results for the polyhedral finite volume method, and the fact that DG is a natural higher-order extension of these schemes, in this work we study some of the properties of DG discretizations on polygonal meshes. To limit the scope, we only investigate the convergence properties of iterative solvers for the discrete systems, assuming an equal number of degrees of freedom per unit area for all element shapes. Future work will also investigate the accuracy of the solutions on the different meshes. We first consider the iterative block Jacobi method applied to a pure convection problem, which in the constant-coefficient case can be solved analytically using von Neumann analysis. Next we apply the solver to Euler's equations of gas dynamics for relevant model flow problems, to obtain numerical results for the convergence of the various element shapes. We consider regular meshes of hexagons, squares, and two different configurations of triangles, as well as the dual of fully unstructured triangular Delaunay refinement meshes. We also perform numerical experiments with the GMRES Krylov subspace solver and a block ILU preconditioner. Although the results are not entirely conclusive, most of the results indicate a clear benefit with the hexagonal and quadrilateral elements over the triangular ones.

The paper is organized as follows. In Section 2, we describe the spatial and the temporal discretizations, and introduce the iterative solvers. In Section 3 we perform the von Neumann analysis of the constant-coefficient advection problem, in 1D and for several mesh configurations in 2D. In Section 4 we show numerical results for more general advection fields, for more general meshes, as well as for

the Euler equations and the GMRES solver. We conclude with a summary of our findings as well as directions for future work.

2. Numerical methods

2.1. ***The discontinuous Galerkin formulation.*** We consider a system of m hyperbolic conservation laws given by the equation

$$\begin{cases} \partial_t \boldsymbol{u} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}) = 0, & (t, \boldsymbol{x}) \in [0, T] \times \Omega, \\ \boldsymbol{u}(0, \boldsymbol{x}) = \boldsymbol{u}_0(\boldsymbol{x}). \end{cases} \tag{1}$$

In order to describe the discontinuous Galerkin spatial discretization, we divide the spatial domain $\Omega \subseteq \mathbb{R}^2$ into a collection of elements, to form the *triangulation* $\mathcal{T}_h = \{K_i\}$. Often the elements K_i are considered to be triangles or quadrilaterals, but in this paper we allow the elements to be arbitrary polygons in order to study the impact of different tessellations on the efficiency of the algorithm.

Let $V_h = \{v_h \in L_2(\Omega) : v_h|_{K_i} \in P^p(K_i)\}$ denote the space of piecewise polynomials of degree p. We let $\boldsymbol{V}_h^m$ denote the space of vector-valued functions of length m, with each component in V_h. Note that continuity is not enforced between the elements. We derive the discontinuous Galerkin method by replacing $\boldsymbol{u}$ in (1) by an approximate solution $\boldsymbol{u}_h \in \boldsymbol{V}_h^m$, and then multiplying equation by a test function $\boldsymbol{v}_h \in \boldsymbol{V}_h^m$. We then integrate by parts over each element. Because the approximate solution $\boldsymbol{u}_h$ is potentially discontinuous at the boundary of an element, the flux function $\boldsymbol{F}$ is approximated by a *numerical flux function* $\widehat{\boldsymbol{F}}$, which takes as arguments $\boldsymbol{u}^+$, $\boldsymbol{u}^-$, and $\boldsymbol{n}$, denoting the solution on the exterior and interior of the element, and the outward-pointing normal vector, respectively. Then, the discontinuous Galerkin method is as follows: find $\boldsymbol{u}_h \in \boldsymbol{V}_h^m$ such that, for all $\boldsymbol{v}_h \in \boldsymbol{V}_h^m$,

$$\int_{K_i} \partial_t \boldsymbol{u}_h \cdot \boldsymbol{v}_h \, dx - \int_{K_i} \boldsymbol{F}(\boldsymbol{u}_h) : \nabla \boldsymbol{v}_h \, dx + \oint_{\partial K_i} \widehat{\boldsymbol{F}}(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n}) \cdot \boldsymbol{v}_h \, ds = 0. \tag{2}$$

2.2. ***Advection equation.*** As a first example, we consider the two-dimensional scalar advection equation

$$u_t + \nabla \cdot (\boldsymbol{\beta} u) = 0, \tag{3}$$

for a given (constant) velocity vector $\boldsymbol{\beta} = (\alpha, \beta)$. We solve this equation in the domain $[0, 2\pi] \times [0, 2\pi]$, with periodic boundary conditions. The exact solution to this equation is given by

$$u(t, x, y) = u_0(x - \alpha t, y - \beta t), \tag{4}$$

where u_0 is the given initial state.

In order to define the discontinuous Galerkin method for (3), we define the *upwind flux* by

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n}) = \begin{cases} \boldsymbol{u}^- & \text{if } \boldsymbol{\beta} \cdot \boldsymbol{n} \geq 0, \\ \boldsymbol{u}^+ & \text{if } \boldsymbol{\beta} \cdot \boldsymbol{n} < 0. \end{cases} \tag{5}$$

We represent the approximate solution function $\boldsymbol{u}_h$ as a vector $\boldsymbol{U}$ consisting of the coefficients of the expansion of $\boldsymbol{u}_h$ in terms of an orthogonal Legendre polynomial modal basis of the function space $\boldsymbol{V}_h^m$. Discretizing (3) results in a linear system of equations, which we can write as

$$\boldsymbol{M}(\partial_t \boldsymbol{U}) + \boldsymbol{L}\boldsymbol{U} = 0, \tag{6}$$

where the mass matrix $\boldsymbol{M}$ corresponds to the first term on the left-hand side of (2), and $\boldsymbol{L}$ consists of the second two terms on the left-hand side. The mass matrix is block-diagonal, and the matrix $\boldsymbol{L}$ is a block matrix, with blocks along the diagonal, and off-diagonal blocks corresponding to the boundary terms from the neighboring elements.

2.3. ***Temporal integration and linear solvers.*** We consider the solution of (6) by means of implicit time integration schemes, the simplest of which is the standard backward Euler scheme,

$$(\boldsymbol{M} + k\boldsymbol{L})\boldsymbol{U}^{n+1} = \boldsymbol{M}\boldsymbol{U}^n. \tag{7}$$

Furthermore, each stage of a higher-order scheme, such as a diagonally implicit Runge–Kutta (DIRK) scheme [1], can be written as a similar equation. The block sparse system can be solved efficiently by means of an iterative linear solver. In this paper, we consider two solvers: the simple block Jacobi method, and the preconditioned GMRES method.

2.3.1. *Block Jacobi method.* A popular and simple iterative solver is the block Jacobi method, defined as follows. Each iteration of the method for solving the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is given by

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{D}^{-1}\boldsymbol{b} + \boldsymbol{R}_J \boldsymbol{x}^{(n)}, \tag{8}$$

where $\boldsymbol{D}$ is the block-diagonal part of $\boldsymbol{A}$, and $\boldsymbol{R}_J = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}$. This simple method has the advantage that it is possible to analyze the convergence properties of the method simply by examining the eigenvalues of the matrix $\boldsymbol{R}_J$. An upper bound of 1 for the absolute value of the eigenvalues of the matrix $\boldsymbol{R}_J$ is a necessary and sufficient condition in order for Jacobi's method to converge (for any choice of initial vector $\boldsymbol{x}^{(0)}$). The spectral radius of $\boldsymbol{R}_J$ determines the speed of convergence.

2.3.2. *Preconditioned GMRES method.* Another popular and oftentimes more efficient [3] method for solving large, sparse linear systems is the GMRES (generalized minimal residual) method [26]. As with most Krylov subspace methods, the choice of preconditioner has a great impact on the efficiency of the solver [24]. A simple and popular choice of preconditioner is the block Jacobi preconditioner. Each application of this preconditioner is performed by multiplying by the inverse of the block-diagonal part of the matrix. Another, often more effective choice of preconditioner is the block ILU(0) preconditioner [8]. This preconditioner produces an approximate blockwise LU factorization, whose sparsity pattern is enforced to be the same as that of the original matrix. This factorization can be performed in place, and requires no more storage than the original matrix. Unlike the block Jacobi method, the block ILU(0) preconditioner can be highly sensitive to the ordering of the mesh elements [11; 4]. Because of this property, it is common to combine the use of ILU preconditioners with certain orderings of the mesh elements designed to increase efficiency, such as reverse Cuthill–McKee [7], minimum degree [20], nested dissection [13], or minimum discarded fill [26].

In this paper, we focus our study on the block Jacobi method, which is simpler and more amenable to analysis. We then perform numerical experiments using both the block Jacobi method and the preconditioned GMRES method using ILU(0) and block Jacobi preconditioning.

3. Jacobi analysis

We compare tessellations of the plane by four sets of *generating patterns*, each consisting of one or more polygons. We consider tessellations consisting of squares, regular hexagons, two right triangles, and two equilateral triangles. The generating patterns considered are shown in Figure 1. Each generating pattern G_j consists of one or two elements, labeled K_j and $\widetilde{K}_j$. We will refer to these generating patterns as S, H, R, and E for squares, hexagons, right triangles, and equilateral triangles.

We are interested in computing the spectral radius of the Jacobi matrix $\boldsymbol{R_J}$ that arises from the discontinuous Galerkin discretization on the mesh resulting

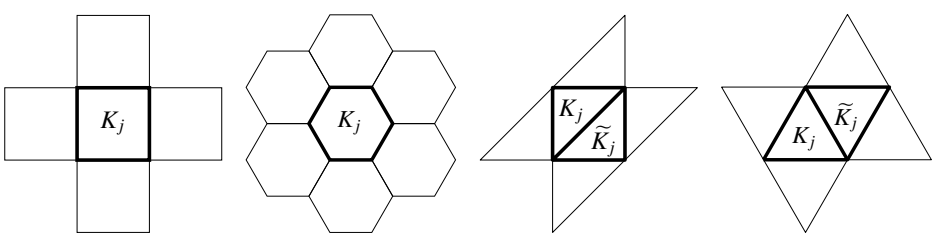


Figure 1. Examples of generating patterns G_j shown with bolded lines. Neighboring elements are shown unbolded. Left to right: square Cartesian grid, regular hexagons, isosceles right triangles, and equilateral triangles.

from tessellating the plane by each of the four generating patterns. For the sake of comparison, we choose the elements from each of the generating patterns to have the same area. Therefore, if the side length of the equilateral triangle is $h_E = h$, then the two equal sides of the isosceles right triangle have side length $h_R = (\sqrt[4]{3}/\sqrt{2})h_E$, the hexagon has side length $h_H = (1/\sqrt{6})h_E$, and the square has side length $h_S = (\sqrt[4]{3}/2)h_E$. Then, the global system will have the same number of degrees of freedom regardless of choice of generating pattern.

3.1. ***Von Neumann analysis.*** First, we compare the efficiency of each of the four types of generating patterns when used to solve the advection equation (3) with the discontinuous Galerkin spatial discretization and implicit time integration. We compute the spectral radius of the matrix $\boldsymbol{R_J}$ using the classical von Neumann analysis for each of the generating patterns, in a manner similar to [16].

Let $\boldsymbol{U}$ denote the solution vector, and let its j-th component, $\boldsymbol{U}_j$, which is itself a vector, denote the degrees of freedom in G_j, the j-th generating pattern. We remark that in the case of squares and hexagons, this corresponds exactly to the degrees of freedom in the element K_j, but in the case of the triangular generating patterns, this corresponds to the degrees of freedom from both of the elements K_j and $\widetilde{K}_j$. In order to determine the eigenvalues of $\boldsymbol{R_J}$, we consider the planar wave with wavenumber (n_x, n_y) defined by

$$\boldsymbol{U}_j = e^{i(n_x x_j + n_y y_j)}\widehat{\boldsymbol{U}}, \tag{9}$$

where (x_j, y_j) are fixed coordinates in G_j. Then, we let ℓ index the generating patterns neighboring G_j, and we let $\boldsymbol{\delta}_\ell = (\delta_{x\ell}, \delta_{y\ell}) = (x_j - x_\ell, y_j - y_\ell)$ be the offsets satisfying $G_j + \boldsymbol{\delta}_\ell = G_\ell$. We can then write the solution in each of the neighboring generating patterns as

$$\boldsymbol{U}_\ell = e^{i(n_x\delta_{x\ell} + n_y\delta_{y\ell})}\boldsymbol{U}_j. \tag{10}$$

In this case we write the semidiscrete equations (6) in the compact form

$$\boldsymbol{M}_j(\partial_t \boldsymbol{U}_j) + \sum_\ell e^{i(n_x\delta_{x\ell} + n_y\delta_{y\ell})}\boldsymbol{L}_{j\ell}\boldsymbol{U}_j = 0, \tag{11}$$

where the summation over ℓ ranges over all neighboring generating patterns, $\boldsymbol{M}_j$ denotes the diagonal block of $\boldsymbol{M}$ corresponding to the j-th generating pattern, and $\boldsymbol{L}_{j\ell}$ denotes the block of $\boldsymbol{L}$ in the j-th row and ℓ-th column. We can write

$$\widehat{\boldsymbol{L}}_j = \sum_\ell e^{i(n_x\delta_{x\ell} + n_y\delta_{y\ell})}\boldsymbol{L}_{j\ell} \tag{12}$$

to further simplify and obtain

$$\boldsymbol{M}_j(\partial_t \widehat{\boldsymbol{U}}) + \widehat{\boldsymbol{L}}_j\widehat{\boldsymbol{U}} = 0. \tag{13}$$

In order to solve (13) using an implicit method, we consider the backward-Euler-type equation

$$(\boldsymbol{M}_j + k\widehat{\boldsymbol{L}}_j)\widehat{\boldsymbol{U}}^{n+1} = \boldsymbol{M}_j\widehat{\boldsymbol{U}}^n. \tag{14}$$

The Jacobi iteration matrix $\boldsymbol{R_J}$ can then be written as

$$\widehat{\boldsymbol{R_J}}_j = \boldsymbol{I} - \boldsymbol{D}^{-1}(\boldsymbol{M}_j + k\widehat{\boldsymbol{L}}_j), \tag{15}$$

where the matrix $\boldsymbol{D} = \boldsymbol{M}_j + k\boldsymbol{L}_{jj}$ consists of the j-th diagonal block of $\boldsymbol{M} + k\boldsymbol{L}$. The eigenvalues of the matrix $\widehat{\boldsymbol{R_J}}_j$ control the speed of convergence of Jacobi's method. In the simple cases of piecewise-constant functions ($p = 0$), or in the case of a one-dimensional domain, the eigenvalues can be computed explicitly. In the more complicated case of $p \geq 1$ in a two-dimensional domain, we compute the eigenvalues numerically.

3.2. ***1D example.*** To illustrate the von Neumann analysis, we consider the one-dimensional scalar advection equation

$$u_t + u_x = 0 \tag{16}$$

on the interval $[0, 2\pi]$ with periodic boundary conditions. We divide the domain into N subintervals K_j, each of length h. Let $\boldsymbol{U}$ denote the solution vector, and let $\boldsymbol{U}_j$ denote the degrees of freedom for the j-th interval K_j. For example, if piecewise constants are used, the method is identical to the upwind finite volume method, and each $\boldsymbol{U}_j$ represents the average of the solution over the interval. If piecewise polynomials of degree p are used, each $\boldsymbol{U}_j$ is a vector of length $p + 1$.

For the purposes of illustration, we choose $p = 1$, and let $\boldsymbol{U}_j = (u_{j,1}, u_{j,2})$ represent the value of the solution at the left and right endpoints of the interval K_j. Then, the local basis on the interval K_j consists of the functions

$$\phi_{j,1}(x) = j - x/h, \qquad \phi_{j,2}(x) = x/h - j + 1. \tag{17}$$

We remark that the upwind flux in this case is always equal to the value of the function immediately to the left of the boundary point:

$$[\widehat{\boldsymbol{F}}(u^+, u^-, x)v(x)]_{(j-1)h}^{jh} = u_{j,2}v_{j,2} - u_{j-1,2}v_{j,1}. \tag{18}$$

The entries of the j-th block of the mass matrix $\boldsymbol{M}$ are given by

$$(\boldsymbol{M}_j)_{i\ell} = \int_{(j-1)h}^{jh} \phi_{j,i}(x)\phi_{j,\ell}(x)\,dx. \tag{19}$$

Additionally, we remark that the diagonal blocks of $\boldsymbol{L}$ consist of the volume integrals and right boundary terms given by

$$(\boldsymbol{L}_{jj})_{i\ell} = \phi_{j,i}(jh)\phi_{i,\ell}(jh) - \int_{(j-1)h}^{jh} \phi'_{j,i}(x)\phi_{j,\ell}(x)\,dx. \tag{20}$$

We let $\boldsymbol{A}$ denote the backward-Euler-type operator defined by

$$\boldsymbol{A} = \boldsymbol{M} + k\boldsymbol{L}, \tag{21}$$

and solving the equation $\boldsymbol{Ax} = \boldsymbol{b}$ by means of Jacobi iterations, we define the Jacobi matrix $\boldsymbol{R_J}$ by

$$\boldsymbol{R_J} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}, \tag{22}$$

where $\boldsymbol{D}$ is the matrix consisting of the diagonal blocks of $\boldsymbol{A}$. The entries of the diagonal blocks $\boldsymbol{M}_j$ and $\boldsymbol{L}_{jj}$ can be computed explicitly using (17) to obtain

$$\boldsymbol{M}_j = \begin{pmatrix} \frac{1}{3}h & \frac{1}{6}h \\ \frac{1}{6}h & \frac{1}{3}h \end{pmatrix}, \qquad \boldsymbol{L}_{jj} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix}, \qquad \boldsymbol{D}_j = \begin{pmatrix} \frac{1}{3}h + \frac{1}{2}k & \frac{1}{6}h + \frac{1}{2}k \\ \frac{1}{6}h - \frac{1}{2}k & \frac{1}{3}h + \frac{1}{2}k \end{pmatrix}. \tag{23}$$

In order to perform the von Neumann analysis, we seek solutions of the form $\boldsymbol{U}_j = e^{inhj}\widehat{\boldsymbol{U}}$, which allows us to explicitly compute the form of the matrix $\widehat{\boldsymbol{L}}_j$. Recalling the compact form from (13), we obtain

$$\widehat{\boldsymbol{L}}_j = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} - e^{-ihn} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix}. \tag{24}$$

Then, the Jacobi matrix $\widehat{\boldsymbol{R_J}}_j$ is given by

$$\widehat{\boldsymbol{R_J}}_j = \begin{pmatrix} 0 & \dfrac{2e^{-ihn}k(2h+3k)}{h^2+4kh+6k^2} \\ 0 & -\dfrac{2e^{-ihn}(h-3k)k}{h^2+4kh+6k^2} \end{pmatrix}, \tag{25}$$

whose eigenvalues λ_1 and λ_2 are given by

$$\lambda_1 = 0, \qquad \lambda_2 = \frac{2k(3k-h)e^{-ihn}}{h^2+4hk+6k^2}. \tag{26}$$

Therefore, each wavenumber n from 0 to $2\pi/h$ corresponds to an eigenvalue of the Jacobi matrix $\boldsymbol{R_J}$, and the magnitude of these eigenvalues determine the speed of convergence of Jacobi's method. In this case, the expression

$$\lambda_{\max} = \frac{2k|h-3k|}{h^2+4hk+6k^2} \tag{27}$$

determines the speed of convergence of Jacobi's method. This expression can easily be seen to be bounded above by 1 for all positive values of h and k, therefore indicating that Jacobi's method is guaranteed to converge, unconditionally, regardless of spatial resolution or time step.

3.3. ***2D analysis.*** We now turn to the analysis of the four generating patterns shown in Figure 1. The analysis proceeds along the same lines as in the one-dimensional example from Section 3.2. As an example, we present the case of piecewise constants, for which it is possible to explicitly compute the eigenvalues of the Jacobi matrix $\boldsymbol{R_J}$. In this case the discontinuous Galerkin formulation simplifies to the upwind finite volume method

$$\int_{K_j} \partial_t u_h \, dx + \oint_{\partial K_j} \widehat{\boldsymbol{F}}(u^+, u^-, \boldsymbol{n}) \, ds = 0. \tag{28}$$

For the sake of concreteness, we assume without loss of generality that the velocity vector $\boldsymbol{\beta} = (\alpha, \beta)$ satisfies $\alpha, \beta \geq 0$. In order to explicitly write the upwind flux on the meshes consisting of hexagons and equilateral triangles, we further assume that $\sqrt{3}\alpha - \beta \geq 0$, and on the mesh consisting of right triangles we assume that $\alpha - \beta \geq 0$. In the case of the square and hexagonal meshes, there is only one degree of freedom per generating pattern, and we will write u_j to represent the average value of the solution over the generating pattern G_j. We then consider the planar wave with wavenumber (n_x, n_y) given by $u_j = e^{i(n_x x_j + n_y y_j)}\hat{u}$. In the case of the square mesh with side length $h_S = (\sqrt[4]{3}/2)h_E$, the method can be written as

$$h_S^2(\partial_t \hat{u}) = -h_S(\alpha(1 - e^{-in_x h_S}) + \beta(1 - e^{-in_y h_S}))\hat{u}. \tag{29}$$

In this case, the mass matrix $\boldsymbol{M}$ is a diagonal matrix with h_S^2 along the diagonal, and the diagonal entries of the matrix $\boldsymbol{L}$ are given by $h_S(\alpha + \beta)$. Therefore, the eigenvalues of the Jacobi matrix $\boldsymbol{R_J^S} = \boldsymbol{I} - D^{-1}(\boldsymbol{M} + k\boldsymbol{L})$ are given by

$$\begin{aligned}\lambda(\boldsymbol{R_J^S}) &= 1 - \frac{1}{h_S^2 + h_S k(\alpha + \beta)}(h_S^2 + h_S k(\alpha(1 - e^{-in_x h_S}) + \beta(1 - e^{-in_y h_S}))) \\ &= \frac{k(\alpha e^{-in_x h_S} + \beta e^{-in_y h_S})}{h_S + k(\alpha + \beta)}.\end{aligned} \tag{30}$$

In the case of the hexagonal mesh with side length $h_H = (1/\sqrt{6})h_E$, the method is

$$\begin{aligned}\frac{3\sqrt{3}}{2}h_H^2(\partial_t \hat{u}) = -h_H\big((\sqrt{3}\alpha + \beta) + \big(-\tfrac{\sqrt{3}}{2}\alpha + \tfrac{\beta}{2}\big)e^{ih_H(-(3/2)n_x + (\sqrt{3}/2)n_y)} \\ + \big(-\tfrac{\sqrt{3}}{2}\alpha - \tfrac{\beta}{2}\big)e^{ih_H(-(3/2)n_x - (\sqrt{3}/2)n_y)} - \beta e^{-ih_H\sqrt{3}n_y}\big)\hat{u}.\end{aligned} \tag{31}$$

A similar analysis shows that the eigenvalues of the matrix $\boldsymbol{R_J^H}$ are given by

$$\begin{aligned}\lambda(\boldsymbol{R_J^H}) = \frac{ke^{-(1/2)ih_H(3n_x + \sqrt{3}n_y)}}{9h_H + 6\alpha k + 2\sqrt{3}\beta k} \times \\ \big(\sqrt{3}\beta(2e^{(1/2)ih_H(3n_x - \sqrt{3}n_y)} - e^{i\sqrt{3}h_H n_y} + 1) + 3\alpha(1 + e^{i\sqrt{3}h_H n_y})\big).\end{aligned} \tag{32}$$

In the case of the two triangular meshes, there are two degrees of freedom per generating pattern, corresponding to the elements K_j and $\widetilde{K}_j$ in the generating pattern G_j. We write $\boldsymbol{U}_j = (u_{j,1}, u_{j,2})$, where $u_{j,1}$ is the average of the solution over the element K_j, and $u_{j,2}$ is the average of the solution over $\widetilde{K}_j$. The planar wave solution is then given by $\boldsymbol{U}_j = e^{i(n_x x_j + n_y y_j)}\widehat{\boldsymbol{U}}$, for $\widehat{\boldsymbol{U}} = (\hat{u}_1, \hat{u}_2)$. We consider the case of a right-triangular mesh, where the two equal sides of the isosceles right triangles have length $h_R = (\sqrt[4]{3}/\sqrt{2})h_E$. The method then reads

$$\partial_t \begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \end{pmatrix} = -\frac{2}{h_R} \begin{pmatrix} \alpha\hat{u}_1 - e^{-ih_R n_x}\alpha\hat{u}_2 \\ \alpha\hat{u}_2 + (\beta-\alpha)\hat{u}_1 - e^{-ih_R n_y}\beta\hat{u}_1 \end{pmatrix}. \tag{33}$$

In the case of the mesh consisting of equilateral triangles, each with side length h_E, the method reads

$$\partial_t \begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \end{pmatrix} = \frac{-4}{\sqrt{3}h_E} \begin{pmatrix} \left(\frac{\sqrt{3}}{2}\alpha + \frac{1}{2}\beta\right)\hat{u}_1 + \left(e^{-ih_E n_x}\left(-\frac{\sqrt{3}}{2}\alpha + \frac{1}{2}\beta\right) - e^{-ih_E n_y}\beta\right)\hat{u}_2 \\ \left(-\frac{\sqrt{3}}{2}\alpha - \frac{1}{2}\beta\right)\hat{u}_1 + \left(\frac{\sqrt{3}}{2}\alpha + \frac{1}{2}\beta\right)\hat{u}_2 \end{pmatrix}. \tag{34}$$

Computing the eigenvalues of the corresponding Jacobi matrices $\boldsymbol{R}_J^{\boldsymbol{R}}$ and $\boldsymbol{R}_J^{\boldsymbol{E}}$, we obtain

$$\lambda(\boldsymbol{R}_J^{\boldsymbol{R}}) = \pm\frac{2ke^{-(1/2)ih_R(n_x+n_y)}\sqrt{\alpha}\sqrt{\beta + (\alpha-\beta)e^{ih_R n_y}}}{h_R + 2\alpha k}, \tag{35}$$

$$\lambda(\boldsymbol{R}_J^{\boldsymbol{E}}) = \pm\frac{2k(3\alpha + \sqrt{3}\beta)\sqrt{2\beta e^{ih_E n_x} + (\sqrt{3}\alpha - \beta)e^{ih_E n_y}}}{(3h_E + 6\alpha k + 2\sqrt{3}\beta k)\sqrt{(\sqrt{3}\alpha + \beta)e^{ih_E(n_x+n_y)}}}. \tag{36}$$

Then, (30), (32), (35), and (36) completely determine the speed of convergence for Jacobi's method of each of the four generating patterns considered. In the case of a higher-order discontinuous Galerkin method with basis consisting of piecewise polynomials of degree $p > 0$, we obtain a Jacobi matrix given by (15), where the matrices $\widehat{\boldsymbol{R}_{\boldsymbol{J}}}_j$, $\boldsymbol{D}$, $\boldsymbol{M}_j$, and $\widehat{\boldsymbol{L}}_j$ are $\frac{1}{2}(p+1)(p+2) \times \frac{1}{2}(p+1)(p+2)$ blocks. In this case, we do not obtain closed-form expressions for the eigenvalues, but rather compute them numerically.

We normalize the velocity magnitude and consider $\boldsymbol{\beta} = (\cos(\theta), \sin(\theta))$. On the square mesh, θ can range from 0 to $\pi/2$. On the hexagonal and equilateral triangle meshes, θ ranges from 0 to $\pi/3$, and on the right-triangular mesh θ ranges from 0 to $\pi/4$. We consider a fixed spatial resolution h, and compare the efficiency of the four patterns for three choices of temporal resolution. We first consider an "explicit" time step, satisfying the CFL-type condition

$$k_{\text{exp}} = \frac{h}{|\boldsymbol{\beta}|}. \tag{37}$$

	$p=0$			$p=1$		
	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**1.000000**	**1.000000**	**1.000000**	**1.000000**	**1.000000**	**1.000000**
squares	1.128939	1.133989	1.136772	1.058098	1.118222	1.130101
right triangles	1.128939	1.133989	1.136772	1.084223	1.132326	1.137313
equilateral triangles	1.207328	1.215467	1.219948	1.137267	1.201638	1.214376
	$p=2$			$p=3$		
hexagons	**1.000000**	**1.000000**	**1.000000**	1.077183	1.070785	1.066101
squares	1.095785	1.118510	1.129314	**1.000000**	**1.000000**	**1.000000**
right triangles	1.111863	1.126951	1.133634	1.010482	1.005391	1.002733
equilateral triangles	1.177503	1.201918	1.213527	1.074570	1.074570	1.074570

Table 1. Ratio of logarithm of eigenvalues $\log \lambda_{\max}(\boldsymbol{R}_J^{\min}) / \log \lambda_{\max}(\boldsymbol{R}_J^*)$ ranging over angle θ and wavenumber (n_x, n_y), for piecewise polynomials of degree 0, 1, 2, and 3, for varying choices of time step k. The smallest eigenvalue in each column is in bold.

As one advantage of using an implicit method is that we are not limited by an explicit time step restriction of the form (37), we consider three implicit time steps given by $k_1 = 3k_{\text{exp}}$, $k_2 = 2k_1$, and $k_3 = 4k_1$. We then maximize over a discrete sample of $\theta \in [0, \pi/4]$ and over all wavenumbers (n_x, n_y), in order to compute the maximum eigenvalue for each of the generating patterns. As the number of iterations required to converge to a given tolerance scales like the reciprocal of the logarithm of the spectral radius, we compare the efficiency of the generating patterns by considering the ratio

$$\frac{\log \lambda_{\max}(\boldsymbol{R}_J^{\min})}{\log \lambda_{\max}(\boldsymbol{R}_J^*)},$$

where $\lambda_{\max}(\boldsymbol{R}_J^*)$ is the largest eigenvalue of $\boldsymbol{R}_J^*$, for $* = H, S, R, E$, and $\lambda_{\max}(\boldsymbol{R}_J^{\min})$ is the smallest among all $\lambda_{\max}(\boldsymbol{R}_J^*)$. This ratio corresponds to the ratio of iterations required to converge to a given tolerance when compared with the most efficient among the generating patterns. The results obtained for $p = 0, 1, 2, 3$, and $k = k_1, k_2, k_3$ for each generating pattern are shown in Table 1 and Figure 2.

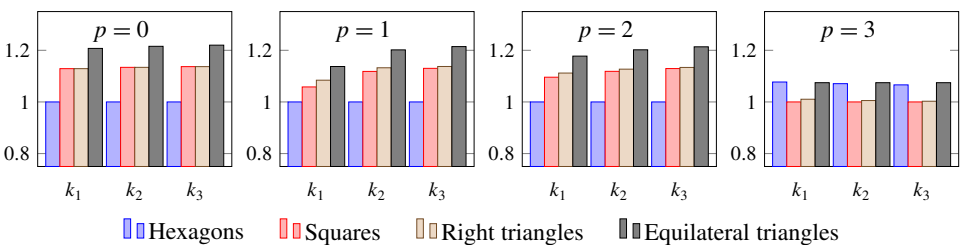


Figure 2. Ratios of the logarithm of the largest eigenvalues for each pattern.

We remark that for polynomials of degree 0, 1, and 2, the hexagonal mesh resulted in the smallest eigenvalues for all choices of time step considered, and the square mesh resulted in the second-smallest eigenvalues. For degree-3 polynomials, the square mesh resulted in the smallest eigenvalues for all cases considered. We notice a significant decrease in the expected performance of the hexagonal elements in the case of $p = 3$, although we have noticed that the effect observed in practice is not as significant as the theoretical results would suggest.

4. Numerical results

4.1. ***Advection with variable velocity field.*** To perform numerical experiments extending the analysis of (3) beyond the case of a constant velocity $\boldsymbol{\beta}$, we consider a variable velocity field $\boldsymbol{\beta}(x, y)$. In this case, the upwind numerical flux

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n}, x, y) = \begin{cases} \boldsymbol{u}^-(x, y) & \text{if } \boldsymbol{\beta}(x, y) \cdot \boldsymbol{n} \geq 0, \\ \boldsymbol{u}^+(x, y) & \text{if } \boldsymbol{\beta}(x, y) \cdot \boldsymbol{n} < 0 \end{cases} \tag{38}$$

is evaluated pointwise. As an example, we define the velocity to be given by the vector field $\boldsymbol{\beta}(x, y) = (2y - 1, -2x + 1)$ on the spatial domain $\Omega = [0, 1] \times [0, 1]$. This velocity field is shown in Figure 3. We let the initial conditions be given by the Gaussian centered at $(x_0, y_0) = (0.35, 0.5)$:

$$u_0(x, y) = \exp(-150((x - x_0)^2 + (y - y_0)^2)). \tag{39}$$

The exact solution is periodic with period π, and is given by the rotation about the center of the domain:

$$u(x, y, t) = \exp\big(-150\big((x - 0.5 + 0.15\cos 2t)^2 + (y - 0.5 - 0.3\cos t \sin t)^2\big)\big). \tag{40}$$

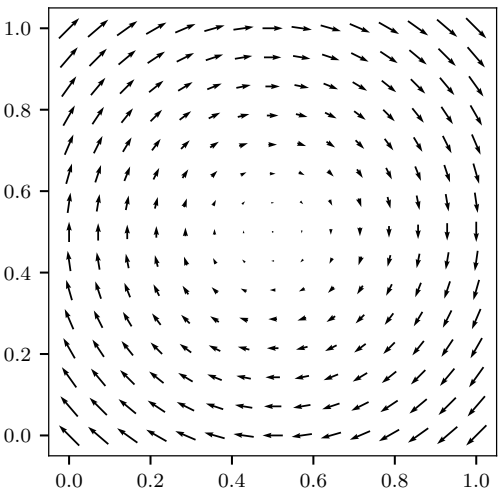


Figure 3. Velocity field $\boldsymbol{\beta}(x, y) = (2y - 1, -2x + 1)$.

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**33**	**57**	**104**	**21**	**41**	**77**	24	**41**	**77**	**21**	**39**	**75**
squares	35	61	109	**21**	42	83	**22**	42	83	22	42	81
right triangles	39	68	128	26	51	100	25	51	100	25	51	100
equilateral triangles	37	67	123	25	47	92	25	47	92	24	47	91

Table 2. Iterations required for the block Jacobi iterative method to converge in the case of a nonconstant velocity field. The smallest number of iterations in each column is in bold.

4.1.1. *Convergence of the block Jacobi method.* We consider meshes of the domain created by repeating each of the four generating patterns considered in the previous section. As before, for fixed spatial resolution h, we choose h_H, h_S, h_R, and h_E such that the number of degrees of freedom is the same for each mesh. We then solve the advection equation using the backward Euler time discretization, where the block Jacobi iterative method is used to solve the resulting linear system. The zero vector is used as the starting vector for the block Jacobi solver. We choose $h = 0.05$, and since $\max_{(x,y)} |\boldsymbol{\beta}(x, y)| = \sqrt{2}$, we consider time steps of $k_1 = h/\sqrt{2}$, $k_2 = 2k_1$, and $k_3 = 4k_1$. The number of iterations required for the block Jacobi method to converge to a tolerance of 10^{-14} are given in Table 2.

The results are similar to those from the analysis performed in Section 3.3. We note that the hexagonal and square meshes resulted in the lowest number of Jacobi iterations for all of the test cases considered. In contrast to the results of Section 3.3, we do not observe a decrease in the performance of the hexagonal elements for the case of $p = 3$, and instead the performance is similar among all choices of p considered.

4.1.2. *Randomly perturbed mesh.* We now consider the effect of polygonal elements on irregular meshes. To this end, we consider a set of *generating points* distributed evenly on a Cartesian grid with mesh size h. Then, each point is perturbed by a random perturbation sampled uniformly from the interval $[-\delta, \delta]$. We obtain two randomized meshes by constructing the Delaunay triangulation and Voronoi diagram resulting from this set of generating points. The Delaunay mesh consists entirely of triangular elements, whereas the Voronoi diagram is constructed out of arbitrary polygonal elements. Examples of the two meshes considered are shown in Figure 4. In contrast to the regular meshes considered in the previous examples, these two meshes do not consist of the same number of elements. The Voronoi diagram consists of about half the number of elements as the Delaunay triangulation. In the test case considered, the randomized polygonal mesh consists of 410 polygonal elements, whereas the randomized triangular mesh consists of 759 triangular elements.

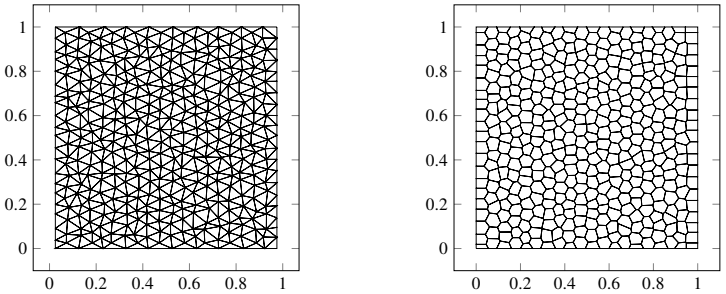


Figure 4. Randomized polygonal and triangular meshes corresponding to the same set of generating points. Left: Delaunay triangulation. Right: Voronoi diagram.

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
Voronoi diagram	**27**	**32**	**38**	**24**	**33**	**38**	**24**	**32**	**36**	**22**	**31**	**36**
Delaunay triangulation	38	48	52	33	45	48	33	46	50	33	44	48

Table 3. Iterations required for the block Jacobi iterative method to converge in the case of irregular, randomly perturbed meshes. The smallest number of iterations in each column is in bold.

The governing equations and setup are the same as in the previous section. We record the number of block Jacobi iterations required to converge to a tolerance of 10^{-14} in Table 3. Because there is a difference in the number of mesh elements, the resulting linear system will have a different total number of degrees of freedom. This difference will then have an additional effect on the speed of convergence of the block Jacobi method. We note that for polynomials of degree $p = 0, 1, 2, 3$ and for all choices of time step k considered, solving the system resulting from the Voronoi diagram requires fewer block Jacobi iterations than does solving the system resulting from the corresponding Delaunay triangulation.

4.1.3. *Convergence of the GMRES method.* The above analysis focused on the block Jacobi method largely because of the simplicity of the method. In practice, more sophisticated iterative methods are often used [26]. In this section, we consider the solution of the linear system (7) by means of the GMRES method, using both the block Jacobi and the block ILU(0) preconditioners. Since the computational work increases per iteration in GMRES, we choose a *restart parameter* of 20 iterations [29]. We repeat the above test case of the advection equation with variable velocity field and record the number of GMRES iterations required to converge to a tolerance of 10^{-14} using the block Jacobi preconditioner in Table 4.

We now consider the solution of the above problem using the GMRES method with the block ILU(0) preconditioner. Because of the sensitivity of the block ILU(0) factorization to the ordering of the mesh elements, and for the sake of a

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**31**	**53**	**92**	**25**	**42**	**80**	28	**47**	**86**	28	**49**	**90**
squares	37	64	116	27	51	101	**27**	51	98	**27**	52	100
right triangles	40	70	134	33	61	123	31	60	117	29	59	115
equilateral triangles	39	67	124	33	58	113	32	59	113	31	57	111

Table 4. Iterations required for the GMRES iterative method with block Jacobi preconditioner to converge. The smallest number of iterations in each column is in bold.

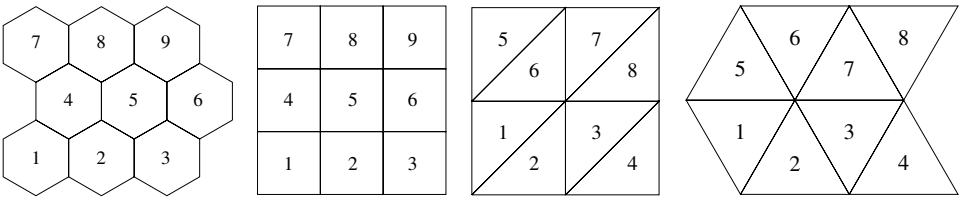


Figure 5. Illustration of the natural ordering of mesh elements. Left to right: square mesh, hexagonal mesh, right-triangular mesh, and equilateral-triangular mesh.

fair comparison between the generating patterns, we consider the *natural ordering* of mesh elements, illustrated in Figure 5. As in the case of the block Jacobi preconditioner, we repeat the test case of the advection equation with variable velocity field. We record the number of GMRES iterations required to converge to the above tolerance using the block ILU(0) preconditioner in Table 5. In this case, the square mesh resulted in the smallest number of iterations in all of the trials. The mesh consisting of right isosceles triangles resulted in the largest number of iterations in all trials. We further note that the number of GMRES iterations required when using the block Jacobi preconditioner scales similarly to the number of block Jacobi iterations required, as recorded in Table 2. We note that the block ILU(0) preconditioner requires fewer GMRES iterations to converge, and the number of iterations scales more favorably in k, when compared with the block Jacobi preconditioner.

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**8**	11	**16**	10	13	20	11	15	23	10	13	22
squares	**8**	**10**	**16**	**8**	**11**	**19**	**7**	**10**	**17**	**8**	**10**	**18**
right triangles	13	19	32	10	14	28	10	15	27	11	14	28
equilateral triangles	11	15	27	10	12	22	9	12	22	9	12	22

Table 5. Iterations required for the GMRES iterative method with ILU(0) preconditioner to converge. The smallest number of iterations in each column is in bold.

4.2. ***Compressible Euler equations.*** The compressible Euler equations of gas dynamics in two dimensions (see, e.g., [14]) are given by

$$\boldsymbol{u}_t + \nabla \cdot \boldsymbol{f}(\boldsymbol{u}) = 0, \tag{41}$$

for

$$\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \qquad \boldsymbol{f}_1(\boldsymbol{u}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho Hu \end{pmatrix}, \qquad \boldsymbol{f}_2(\boldsymbol{u}) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho Hv \end{pmatrix}, \tag{42}$$

where ρ is the density, $\boldsymbol{v} = (u, v)$ is the fluid velocity, p is the pressure, and E is the specific energy. The total enthalpy H is given by

$$H = E + \frac{p}{\rho}, \tag{43}$$

and the pressure is determined by the equation of state

$$p = (\gamma - 1)\rho(E - \tfrac{1}{2}\boldsymbol{v}^2), \tag{44}$$

where $\gamma = c_p/c_v$ is the ratio of specific heat capacities at constant pressure and constant volume.

We consider the model problem of an unsteady compressible vortex in a rectangular domain [32]. The domain is taken to be a 20×15 rectangle, and the vortex is initially centered at $(x_0, y_0) = (5, 5)$. The vortex is moving with the free stream at an angle of θ. The exact solution is given by

$$u = u_\infty \left(\cos(\theta) - \frac{\epsilon((y - y_0) - \bar{v}t)}{2\pi r_c} \exp\left(\frac{f(x, y, t)}{2} \right) \right), \tag{45}$$

$$u = u_\infty \left(\sin(\theta) - \frac{\epsilon((x - x_0) - \bar{u}t)}{2\pi r_c} \exp\left(\frac{f(x, y, t)}{2} \right) \right), \tag{46}$$

$$\rho = \rho_\infty \left(1 - \frac{\epsilon^2(\gamma - 1)M_\infty^2}{8\pi^2} \exp(f(x, y, t)) \right)^{1/(\gamma-1)}, \tag{47}$$

$$p = p_\infty \left(1 - \frac{\epsilon^2(\gamma - 1)M_\infty^2}{8\pi^2} \exp(f(x, y, t)) \right)^{\gamma/(\gamma-1)}, \tag{48}$$

where $f(x, y, t) = (1 - ((x - x_0) - \bar{u}t)^2 - ((y - y_0) - \bar{v}t)^2)/r_c^2$, M_∞ is the Mach number, and u_∞, ρ_∞, and p_∞ are the free-stream velocity, density, and pressure, respectively. The free-stream velocity is given by $(\bar{u}, \bar{v}) = u_\infty(\cos(\theta), \sin(\theta))$. The strength of the vortex is given by ϵ, and its size is r_c. We choose the parameters to be $\gamma = 1.4$, $M_\infty = 0.5$, $u_\infty = 1$, $\theta = \arctan(\frac{1}{2})$, $\epsilon = 0.3$, and $r_c = 1.5$.

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**32**	**49**	**78**	**31**	**50**	**83**	**50**	**90**	**158**	**53**	**97**	**171**
squares	34	51	89	**31**	54	92	54	99	181	55	105	201
right triangles	37	56	97	41	64	112	58	101	189	59	113	217
equilateral triangles	37	57	95	39	62	113	54	99	179	60	114	215

Table 6. Block Jacobi iterations required per Newton solve of the compressible Euler equations. The lowest number of iterations in each column is in bold.

In the discontinuous Galerkin discretization of the Euler equations we use the Lax–Friedrichs numerical flux defined by

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n}) = \tfrac{1}{2}(\boldsymbol{f}(\boldsymbol{u}^-)\cdot\boldsymbol{n} + \boldsymbol{f}(\boldsymbol{u}^+)\cdot\boldsymbol{n} + \alpha(\boldsymbol{u}^- - \boldsymbol{u}^+)), \tag{49}$$

where α is the maximum absolute eigenvalue over $\boldsymbol{u}^-$ and $\boldsymbol{u}^+$ of the matrix $B(\boldsymbol{u}, \boldsymbol{n})$ defined by

$$B(\boldsymbol{u}, \boldsymbol{n}) = J_{f_1} n_1 + J_{f_2} n_2, \tag{50}$$

where J_{f_1} and J_{f_2} are the Jacobian matrices of the components of the numerical flux function $\boldsymbol{f}$ defined in (42).

We use the backward Euler time discretization, but remark that (2) results in a nonlinear set of equations, which is solved using Newton's method. Each iteration of Newton's method requires solving a linear equation of the form (7). We set $h = 1$, and consider three time steps: $k_1 = 0.03h$, $k_2 = 2k_1$, and $k_3 = 4k_1$. We use piecewise polynomials of degrees $p = 0, 1, 2, 3$. Each Newton solve requires between 3 and 8 iterations to converge within a tolerance of 5×10^{-13}. The tolerance used for the linear solvers is the same as in the previous test cases.

4.2.1. *The block Jacobi method.* Each iteration of Newton's method requires the solution of a linear system of equations. We solve these systems using the block Jacobi method. We compute the total number of Jacobi iterations required to complete one solve of Newton's method, and report the results in Table 6. We note that for each choice of p and time step k, the hexagonal mesh required the lowest number of block Jacobi iterations. As in the previous numerical experiments, we do not see a decrease in performance for the hexagonal elements in the case of $p = 3$. The square mesh resulted in the second-smallest number of iterations for most of the cases considered, while the two configurations of triangles resulted in generally similar numbers of iterations.

4.2.2. *The GMRES method.* We now repeat the above test case, using the GMRES method to solve the resulting linear systems. We consider both the block Jacobi and block ILU(0) preconditioners. We then compute the total number of GMRES iterations required to complete one solve of Newton's method. As in Section 4.1.3,

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**55**	**74**	**106**	**50**	**92**	**126**	**61**	**110**	**153**	**76**	**141**	**195**
squares	62	84	155	52	93	132	67	126	185	78	149	222
right triangles	63	87	162	81	106	184	96	132	242	85	159	299
equilateral triangles	66	90	167	81	108	187	72	133	197	85	161	245

Table 7. GMRES with block Jacobi preconditioner: iterations required per Newton solve of the compressible Euler equations. The lowest number in each column is in bold.

	$p=0$			$p=1$			$p=2$			$p=3$		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
hexagons	**24**	32	**42**	**21**	36	48	29	48	57	29	50	64
squares	**24**	**28**	45	**21**	**33**	**40**	**24**	**41**	**49**	**27**	**48**	**60**
right triangles	31	40	70	35	40	60	36	48	69	31	49	75
equilateral triangles	28	37	65	37	44	70	33	56	68	38	64	80

Table 8. GMRES with block ILU(0) preconditioner: iterations required per Newton solve of the compressible Euler equations. The lowest number in each column is in bold.

the ordering of the mesh elements has a significant effect on the effectiveness of the block ILU(0) approximate factorization. For this reason, we use the natural ordering of elements, depicted in Figure 5. We present the results for the block Jacobi preconditioner in Table 7, and for the block ILU(0) preconditioner in Table 8. With the block Jacobi preconditioner, the hexagonal mesh required the smallest number of iterations for all test cases considered, and the square mesh the second-smallest. In the case of the block ILU(0) preconditioner, the square mesh required the lowest number of iterations, with the hexagonal mesh usually requiring the second-smallest number of iterations. As we observed in Section 4.1.3, the number of iterations required for both the block Jacobi method and GMRES with the block Jacobi preconditioner scales quite poorly with increasing time steps. The number of GMRES iterations required when using the block ILU(0) preconditioner is significantly better.

4.3. ***Inviscid flow problems.*** The following two numerical experiments extend the above results to larger-scale, more realistic flow problems. These problems, in contrast to the preceding test cases, are characterized by a large number of degrees of freedom, the presence of geometric features and wall boundary conditions, variably sized mesh elements, and shocks. As in the previous section, the equations considered here are the compressible Euler equations. For the following two problems, we choose the finite element function space to consist of piecewise-constant functions (corresponding to $p = 0$), which results in a finite-volume-type

discretization. This choice of discretization allows for the solution of problems with shocks, without the use of slope limiters, artificial viscosity, or other shock-capturing techniques [17]. The Roe numerical flux is used as an approximate Riemann solver for these problems.

4.3.1. *Subsonic flow over a circular cylinder.* For a first test case, we consider the inviscid flow over a circular cylinder at Mach 0.2. The computational domain is defined as $\Omega = R \setminus C$, where $R = [-10, 30] \times [-10, 20]$, and C is a disk of radius 1 centered at the point (5, 5). Far-field boundary conditions are enforced on ∂R, and a no-normal-flow condition is enforced on ∂C. The free-stream velocity is taken to be unity in the x-direction, and $\rho_\infty = 1$.

For this test case we use four unstructured meshes, two consisting entirely of triangles and two consisting of mixed polygons, generated using the PolyMesher algorithm [31]. All the meshes are created using a gradient-limited element size function that determines the initial distribution of seed points according to the rejection method [25], such that the element edge length near the surface of the cylinder is about one-fifth the edge length of elements away from the cylinder. For both the triangular and polygonal meshes, we consider a coarse mesh, with 15,404 elements, and a fine mesh with 62,270 elements. Thus, the average area of each element is the same for both the polygonal and triangular meshes. Additionally, the number of degrees of freedom in the solution is the same, allowing for a fair comparison. The coarse polygonal mesh and a zoom-in around the surface of the cylinder are shown in Figure 6.

Starting from free-stream initial conditions, we integrate the equations until $t = 5 \times 10^{-3}$ in order to obtain a representative solution. Using this solution, we then compute 10 time steps using a third-order A-stable DIRK method [1]. Each stage of the DIRK method requires the solution of a nonlinear system of equations,

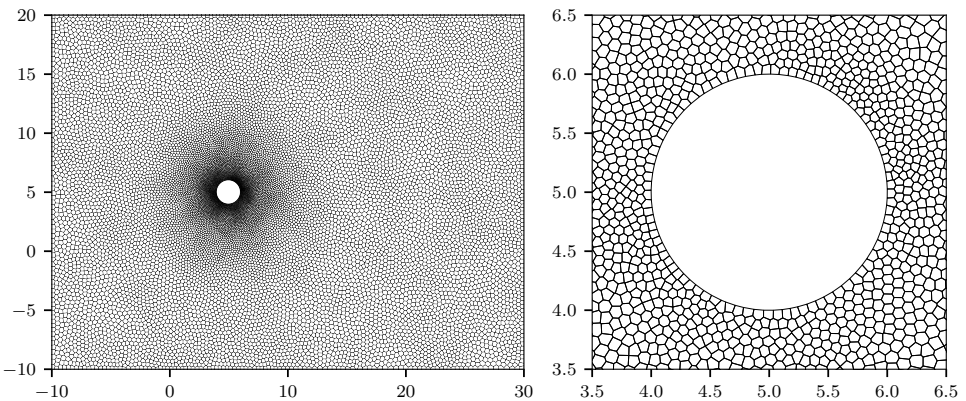


Figure 6. Overview of the coarse mesh with 15,404 elements, with zoom-in showing polygonal elements near the surface of the cylinder.

Δt	ILU		Jacobi		ratios	
	polygonal	triangular	polygonal	triangular	ILU	Jacobi
1.0×10^{-1}	793	932	2092	3126	0.85	0.67
2.5×10^{-1}	1569	1829	4405	6870	0.86	0.64
5.0×10^{-1}	2470	3090	7145	11859	0.80	0.60
1.0	3651	4486	11054	18880	0.81	0.59
1.0×10^{-1}	1443	1673	4075	6137	0.86	0.66
2.5×10^{-1}	2998	3344	8732	12741	0.90	0.69
5.0×10^{-1}	4720	5423	14084	21882	0.87	0.64
1.0	7205	8151	22814	34706	0.88	0.66

Table 9. Total GMRES iterations per 10 time steps for inviscid flow over a circular cylinder. Top: coarse grid with 15,404 elements. Bottom: fine mesh with 95,932 elements.

Δt	polygonal	triangular	ratio	polygonal	triangular	ratio
1.0×10^{-1}	2474	3159	0.78	4788	6281	0.76
2.5×10^{-1}	4895	6697	0.73	9609	12406	0.77
5.0×10^{-1}	7882	12158	0.65	15580	20946	0.74
1.0	13181	19072	0.69	26628	33934	0.78

Table 10. Total block Jacobi iterations per 10 time steps for inviscid flow over a circular cylinder. Left: coarse grid with 15,404 elements. Right: fine mesh with 95,932 elements.

which we solve by means of Newton's method. In each iteration of Newton's method, we solve the resulting linear system of the form (7) using both the block Jacobi method and the preconditioned GMRES method. The nonlinear system is solved to within a tolerance of 10^{-8}, and each linear system is solved using a relative tolerance of 10^{-5}. For the GMRES method, we consider two preconditioners: block Jacobi, and block ILU(0). In order to compare the iterative solver performance differences between meshes, we compute the total number of solver iterations required to complete all 10 time steps. The results for the GMRES method are shown in Table 9, and for the block Jacobi solver in Table 10.

These results demonstrate a consistent trend, corroborating both the numerical results and the analysis from the previous sections. When using the block Jacobi solver or GMRES with block Jacobi preconditioner, the polygonal mesh results in convergence in 60–70% of the iterations required for the triangular mesh. The effect is smaller when using the ILU(0) preconditioner, but we do still observe a modest reduction in the number of iterations required. When using the block Jacobi iterative solver, we observe iteration counts very similar to when using GMRES with block Jacobi as a preconditioner. In these cases, the polygonal mesh requires 70–80% of the iterations as the all-triangular mesh.

4.3.2. *Supersonic flow over a circular cylinder.* The next numerical example is designed to investigate the performance of the iterative solvers for steady-state problems, in the presence of shocks and h-adapted meshes. For this problem, we let the domain be $\Omega = R \setminus C$, where $R = [0, 5] \times [0, 10]$ and, as before, C is a circle of radius 1 centered at (5, 5). Free-stream conditions are enforced at the left, top, and bottom boundaries, an inviscid wall condition is enforced on the boundary of the cylinder, and an outflow condition is enforced on the right boundary. The Mach number is set to $M = 2.0$, resulting in the formation of a shock upstream from the cylinder. In order to accurately capture the shock, we refine the mesh in its vicinity. As in the previous case, we consider a set of four meshes: two all-triangular and two polygonal. For both the triangular and polygonal meshes, we consider coarse and fine versions, with 31,162 and 95,932 elements, respectively. The coarse mesh is depicted in Figure 7, left, with Mach isolines overlaid to indicate the position of the shock. Additionally, Mach contours of the steady-state solution are shown in Figure 7, right.

Beginning with free-stream initial conditions, the solution rapidly approaches a steady state. We integrate in time until $t = 100$ in order to obtain a solution which can be used as an initial guess for the steady-state Newton solve. Then, starting with this solution, we set the time derivative of the solution to zero and solve the resulting nonlinear equations using Newton's method to find a steady-state solution. The

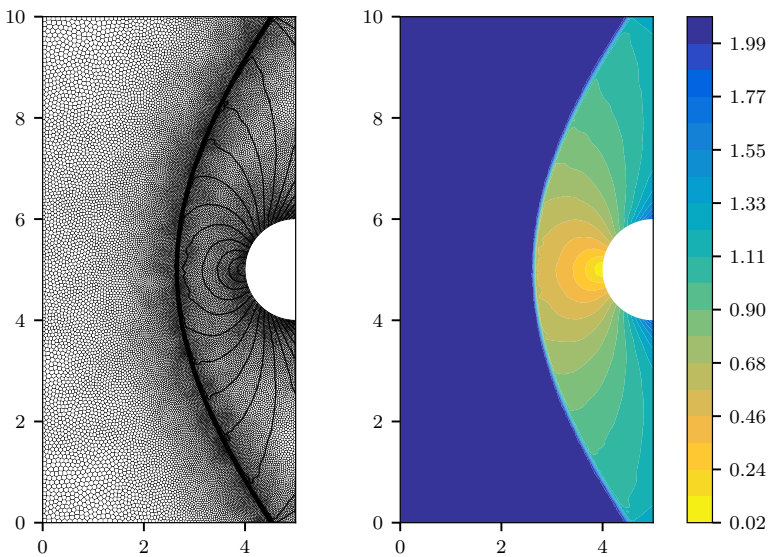


Figure 7. Overview of coarse polygonal mesh with 31,162 elements, showing Mach number contours for steady-state solution. Left: coarse mesh for supersonic test problem, showing Mach isolines for steady-state solution. Right: contours of Mach number for steady-state solution.

	polygonal	triangular	ratio	polygonal	triangular	ratio
ILU	469	640	0.73	953	1947	0.49
Jacobi	2340	6464	0.36	–	–	–

Table 11. Total GMRES iterations per steady-state solve for supersonic flow over a cylinder. Left: coarse grid with 31,162 elements. Right: fine mesh with 95,932 elements.

resulting linear system that is required to be solved at each iteration can be thought of as corresponding to (7), where formally we set $k = \infty$. The nonlinear system is solved to within a tolerance of 10^{-10}, and each linear system is solved using a relative tolerance of 10^{-5}. Since the mass matrix in (7) acts to regularize the linear system, the conditioning becomes worse for larger values of k, and the number of iterations required per linear solve grows. Hence, effective preconditioners are particularly important for the solution of such steady-state problems. For these problems, the block Jacobi iterative solver did not converge in fewer than 10,000 iterations, and so we consider only the GMRES method, using block ILU(0) and block Jacobi preconditioners.

We present the comparison of iteration counts for this problem in Table 11. On the coarse meshes, the ILU(0) preconditioner required about 73% as many iterations on the polygonal mesh when compared with the triangular mesh. This difference is more significant when using the block Jacobi preconditioner, consistent with the results observed in previous sections. In this case, the polygonal mesh requires only slightly more than one third the number of iterations as the all-triangular mesh. On the fine mesh, there are close to half a million degrees of freedom. For a problem of this scale, we did not observe convergence in fewer than 10,000 iterations per linear solve using the block Jacobi preconditioner, and so we only compare performance using the block ILU(0) preconditioner. In this case, the polygonal mesh required about half as many iterations per steady-state solve when compared with the all-triangular mesh.

5. Conclusions

In this paper we have analyzed the effect of the generating pattern of a regular mesh on the convergence of iterative linear solvers applied to implicit discontinuous Galerkin discretizations. We considered four generating patterns: a hexagon, a square, two right triangles, and two equilateral triangles.

A classical von Neumann analysis applied to the constant-velocity advection equation allowed us to compute the eigenvalues of the block Jacobi matrix, and therefore estimate the speed of convergence of the block Jacobi method. In more than half of the cases considered, the hexagonal generating pattern resulted in the smallest eigenvalues, and in the remaining cases, the square generating pattern

resulted in the smallest eigenvalues.

In order to extend these results beyond the case of the constant-velocity advection equation, we performed numerical experiments on the variable-velocity advection equation and compressible Euler equations. In the case of the advection equation, in all but one case the hexagonal mesh resulted in the fastest convergence, and in the remaining case the square mesh resulted in the fastest convergence. In the case of the Euler equations, the hexagonal mesh resulted in the fastest convergence in all test cases.

We additionally considered two irregular meshes resulting from the random perturbation of a set of regularly spaced generating points. We obtain a triangular mesh by performing the Delaunay triangulation on these points, and we obtain a polygonal mesh by constructing the Voronoi diagram dual to the Delaunay triangulation. Solving the advection equation on these irregular meshes, we observed that the block Jacobi method converged faster on the polygonal mesh in every test case. Additionally, we performed numerical experiments examining the performance of the GMRES iterative method when used with the ILU(0) preconditioner. We found that in all of the test cases, the square generating pattern resulted in the lowest number of GMRES iterations, and in all but two cases, the hexagonal generating pattern resulted in the second-lowest number of iterations.

For a final set of numerical experiments, we performed two inviscid fluid flow simulations on sets of coarse and fine meshes. Each mesh was either all-triangular, or was composed of arbitrary polygons. We measured iteration counts for both time-dependent and steady-state problems, using the block Jacobi method, and GMRES with block ILU(0) and block Jacobi preconditioners. We found that the polygonal meshes resulted in faster convergence of the iterative solvers, with a larger difference being observed for the block Jacobi method and preconditioner. This difference was more pronounced for the steady-state problem, with quite a significant difference observed on the fine mesh using GMRES with ILU(0).

These results suggest that certain types of polygonal meshes have the advantage of rapid convergence of iterative solvers. Future research directions involve the study of accuracy of DG methods on polygonal and polyhedral meshes, efficient computation of quadrature rules over arbitrary polygonal domains, and the extension of the above results to three spatial dimensions.

Acknowledgements

This work was supported by the AFOSR Computational Mathematics program under grant number FA9550-15-1-0010. Pazner was supported by the Department of Defense through the National Defense Science and Engineering Graduate Fellowship Program and by the Natural Sciences and Engineering Research Council of Canada.

References

[1] R. Alexander, *Diagonally implicit Runge–Kutta methods for stiff O.D.E.'s*, SIAM J. Numer. Anal. **14** (1977), no. 6, 1006–1021. MR Zbl

[2] G. Balafas, *Polyhedral mesh generation for CFD-analysis of complex structures*, master's thesis, Technische Universität München, 2014.

[3] F. Bassi and S. Rebay, *GMRES discontinuous Galerkin solution of the compressible Navier–Stokes equations*, Discontinuous Galerkin methods (B. Cockburn, G. E. Karniadakis, and C.-W. Shu, eds.), Lect. Notes Comput. Sci. Eng., no. 11, Springer, 2000, pp. 197–208. MR Zbl

[4] M. Benzi, W. Joubert, and G. Mateescu, *Numerical experiments with parallel orderings for ILU preconditioners*, Electron. Trans. Numer. Anal. **8** (1999), 88–114. MR Zbl

[5] M. Berggren, *A vertex-centered, dual discontinuous Galerkin method*, J. Comput. Appl. Math. **192** (2006), no. 1, 175–181. MR Zbl

[6] B. Cockburn and C.-W. Shu, *Runge–Kutta discontinuous Galerkin methods for convection-dominated problems*, J. Sci. Comput. **16** (2001), no. 3, 173–261. MR Zbl

[7] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, ACM '69: proceedings of the 1969 24th National Conference, Association for Computing Machinery, 1969, pp. 157–172.

[8] L. T. Diosady and D. L. Darmofal, *Preconditioning methods for discontinuous Galerkin solutions of the Navier–Stokes equations*, J. Comput. Phys. **228** (2009), no. 11, 3917–3935. MR Zbl

[9] B. Diskin and J. L. Thomas, *Comparison of node-centered and cell-centered unstructured finite-volume discretizations: inviscid fluxes*, AIAA J. **49** (2011), no. 4, 836–854.

[10] B. Diskin, J. L. Thomas, E. J. Nielsen, H. Nishikawa, and J. A. White, *Comparison of node-centered and cell-centered unstructured finite-volume discretizations: viscous fluxes*, AIAA J. **48** (2010), no. 7, 1326–1338.

[11] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT **29** (1989), no. 4, 635–657. MR Zbl

[12] R. V. Garimella, J. Kim, and M. Berndt, *Polyhedral mesh generation and optimization for non-manifold domains*, Proceedings of the 22nd International Meshing Roundtable (J. Sarrate and M. Staten, eds.), Springer, 2014, pp. 313–330.

[13] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal. **10** (1973), 345–363. MR Zbl

[14] R. Hartmann, *Discontinuous Galerkin methods for compressible flows: higher order accuracy, error estimation and adaptivity*, 34th CFD: higher order discretization methods (H. Deconinck and M. Ricchiuto, eds.), Von Karman Institute Lecture Series, no. 2006-01, Von Karman Institute, 2006.

[15] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, Texts in Applied Mathematics, no. 54, Springer, 2008. MR Zbl

[16] E. J. Kubatko, C. Dawson, and J. J. Westerink, *Time step restrictions for Runge–Kutta discontinuous Galerkin methods on triangular grids*, J. Comput. Phys. **227** (2008), no. 23, 9697–9710. MR Zbl

[17] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, Cambridge University, 2002. MR Zbl

[18] H. Luo, J. D. Baum, and R. Löhner, *A discontinuous Galerkin method based on a Taylor basis for the compressible flows on arbitrary grids*, J. Comput. Phys. **227** (2008), no. 20, 8875–8893. MR Zbl

[19] G. Manzini, A. Russo, and N. Sukumar, *New perspectives on polygonal and polyhedral finite element methods*, Math. Models Methods Appl. Sci. **24** (2014), no. 8, 1665–1699. MR Zbl

[20] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Sci. **3** (1957), 255–269. MR Zbl

[21] W. Oaks and S. Paoletti, *Polyhedral mesh generation*, 9th International Meshing Roundtable, Sandia National Laboratories, 2000, pp. 57–67.

[22] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz, *Adaptive remeshing for compressible flow computations*, J. Comput. Phys. **72** (1987), no. 2, 449–466. Zbl

[23] M. Peric, *Flow simulation using control volumes of arbitrary polyhedral shape*, ERCOFTAC Bull. **62** (2004), 25–29.

[24] P.-O. Persson and J. Peraire, *Newton–GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations*, SIAM J. Sci. Comput. **30** (2008), no. 6, 2709–2733. MR Zbl

[25] P.-O. Persson, *Mesh generation for implicit geometries*, Ph.D. thesis, Massachusetts Institute of Technology, 2005. MR

[26] _____, *Scalable parallel Newton–Krylov solvers for discontinuous Galerkin discretizations*, 47th AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, 2009.

[27] W. H. Reed and T. R. Hill, *Triangular mesh methods for the neutron transport equation*, technical report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[28] J. Ruppert, *A Delaunay refinement algorithm for quality 2-dimensional mesh generation*, J. Algorithms **18** (1995), no. 3, 548–585. MR Zbl

[29] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed., Society for Industrial and Applied Mathematics, 2003. MR Zbl

[30] J. R. Shewchuk, *Delaunay refinement algorithms for triangular mesh generation*, Comput. Geom. **22** (2002), no. 1–3, 21–74. MR Zbl

[31] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes, *PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab*, Struct. Multidiscip. Optim. **45** (2012), no. 3, 309–328. MR Zbl

[32] Z. J. Wang, K. Fidkowski, R. Abgrall, and et al., *High-order CFD methods: current status and perspective*, Internat. J. Numer. Methods Fluids **72** (2013), no. 8, 811–845. MR

Received August 8, 2016. Revised September 25, 2017.

WILL PAZNER: will_pazner@brown.edu
Division of Applied Mathematics, Brown University, Providence, RI, United States

PER-OLOF PERSSON: persson@berkeley.edu
Department of Mathematics, University of California, Berkeley, Berkeley, CA, United States

Communications in Applied Mathematics and Computational Science

msp.org/camcos

See inside back cover or msp.org/camcos for submission instructions.

The subscription price for 2018 is US $100/year for the electronic version, and $150/year (+$15, if shipping outside the US) for print and electronic. Subscriptions, requests for back issues from the last three years and changes of subscriber address should be sent to MSP.

Communications in Applied Mathematics and Computational Science (ISSN 2157-5452 electronic, 1559-3940 printed) at Mathematical Sciences Publishers, 798 Evans Hall #3840, c/o University of California, Berkeley, CA 94720-3840, is published continuously online. Periodical rate postage paid at Berkeley, CA 94704, and additional mailing offices.

CAMCoS peer review and production are managed by EditFLOW® from MSP.

PUBLISHED BY

mathematical sciences publishers

nonprofit scientific publishing

http://msp.org/

Communications in Applied Mathematics and Computational Science

vol. 13 no. 1 2018