# A Sparse and High-Order Accurate Line-Based Discontinuous Galerkin Method for Unstructured Meshes

Per-Olof Persson[a,*]

[a]*Department of Mathematics, University of California, Berkeley, Berkeley, CA 94720-3840, USA*

**Abstract**

We present a new line-based discontinuous Galerkin (DG) discretization scheme for first- and second-order systems of partial differential equations. The scheme is based on fully unstructured meshes of quadrilateral or hexahedral elements, and it is closely related to the standard nodal DG scheme as well as several of its variants such as the collocation-based DG spectral element method (DGSEM) or the spectral difference (SD) method. However, our motivation is to maximize the sparsity of the Jacobian matrices, since this directly translates into higher performance in particular for implicit solvers, while maintaining many of the good properties of the DG scheme. To achieve this, our scheme is based on applying one-dimensional DG solvers along each coordinate direction in a reference element. This reduces the number of connectivities drastically, since the scheme only connects each node to a line of nodes along each direction, as opposed to the standard DG method which connects all nodes inside the element and many nodes in the neighboring ones. The resulting scheme is similar to a collocation scheme, but it uses fully consistent integration along each 1-D coordinate direction which results in different properties for nonlinear problems and curved elements. Also, the scheme uses solution points along each element face, which further reduces the number of connections with the neighboring elements. Second-order terms are handled by an LDG-type approach, with an upwind/downwind flux function based on a switch function at each element face. We demonstrate the accuracy of the method and compare it to the standard nodal DG method for problems including Poisson's equation, Euler's equations of gas dynamics, and both the steady-state and the transient compressible Navier-Stokes equations. We also show how to integrate the Navier-Stokes equations using implicit schemes and Newton-Krylov solvers, without impairing the high sparsity of the matrices.

*Keywords:* High-order, sparse, discontinuous Galerkin, unstructured meshes, Navier-Stokes

## 1. Introduction

In recent years it has become clear that the current computational methods for scientific and engineering phenomena are inadequate for challenging problems. These include problems with propagating waves, turbulent fluid flow, nonlinear interactions, and multiple scales. This has resulted in a significant interest in so-called high-order accurate methods, which have the potential to produce fundamentally more reliable solutions. A number of numerical methods have been proposed, including multi-block finite difference methods [1, 2, 3], high-order finite volume methods [4, 5], stabilized finite element methods [6], discontinuous Galerkin (DG) methods [7, 8, 9], DG spectral element methods (DGSEM) [10], spectral volume/difference methods [11, 12, 13, 14], and hybridized DG methods [15, 16]. All methods have advantages in particular situations, but for various reasons most general purpose commercial-grade simulation tools still use traditional low-order methods.

*Corresponding author. Tel.: +1-510-642-6947; Fax.: +1-510-642-8204.
*Email address:* persson@berkeley.edu (Per-Olof Persson)

Much of the current research is devoted to the discontinuous Galerkin method. This is partly because of its many attractive properties, including the use of fully unstructured simplex meshes, the natural stabilization mechanism based on approximate Riemann solvers, and the rigorous theoretical foundations. It can certainly be discussed why the DG method is not used routinely for real-world simulations, but one of the main reasons is clearly its high computational cost, which is still at least a magnitude more than low-order methods or high-order finite difference methods on similar grids. For some problems, explicit time-stepping or matrix-free implicit methods can be employed, but for many real-world problems and meshes full Jacobian matrices are required for the solvers to be efficient. Here, nodal-based Galerkin methods have a fundamental disadvantage in that they connect all unknowns inside an element, as well as all neighboring face nodes, even for first-order derivatives. This leads to a stencil size that scales like $p^D$ for polynomial degrees $p$ in $D$ spatial dimensions. As a contrast, a standard finite difference method only connects neighboring nodes along the $D$ coordinate lines through the node. This gives a stencil size proportional to $Dp$, which in three dimensions can be magnitudes smaller ever for moderate values of $p$.

Several high-order schemes for unstructured meshes have been proposed with a similar stencil-size reduction. In particular, the DG spectral element method [10, 17] is a collocation-based method on a staggered grid which only uses information along each coordinate line for the discretized equations. Other closely related schemes have the same property, such as the spectral difference method [12], the flux reconstruction method [13, 14], and the DGM-FD method [18]. For the special case of a linear one-dimensional problem, many of these methods can be shown to be identical to the standard DG method [19], but in general they define different schemes with varying properties.

In an attempt to further reduce the size of the Jacobians, and to ensure that the scheme is identical to the standard DG method along each line of nodes, we propose a new line-based DG scheme. Like the DGSEM, our Line-DG scheme is derived by considering only the 1-D problems that arise along each coordinate direction. We apply standard 1-D DG formulations for each of these sub-problems, and all integrals are computed fully consistently (with sufficient accuracy), which means in particular that the definition of the scheme makes no statement about flux points. We note that this can be done without introducing additional connectivities, since all nodes in the local 1-D problem are already connected by the shape functions. In addition, our scheme uses solution points along each element face, which further reduces the number of connectivities with the neighboring elements.

For the second-order terms in the Navier-Stokes equations, we use an LDG-type approach [20] with upwind/downwind fluxes based on consistent switches along all globally connected lines of elements. Special care is required to preserve the sparsity of the resulting matrices, and we propose a simple but efficient Newton-Krylov solver which splits the matrix product in order to avoid introducing additional matrix entries. Many options for preconditioning are possible, and in this work we use a block-Jacobi method with sparse blocks.

We first describe the method for first-order systems in Section 2 and mention some practical implementation issues, including a study of the structure of the Jacobian matrices. In Section 3 we extend the scheme to second-order systems using the LDG-type scheme, and in Section 4 we discuss the implicit temporal discretization, some approaches for maintaining the high sparsity of the discretization, and the Newton-Krylov solver. Finally, in Section 5 we show numerical results and convergence for Poisson's equation, an inviscid Euler vortex, and flow over a cylinder. We also compare the method to the standard nodal DG method, and we conclude that the differences are overall very small. For the Navier-Stokes equations, we show convergence of drag and lift forces for steady-state laminar flow around an airfoil, and we demonstrate our implicit time-integrators on a transient LES-type flow problem.

## 2. Line-based discontinuous Galerkin discretization

### *2.1. First-order equations*

Consider a system of $m$ first-order conservation laws with source terms,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}) = \boldsymbol{S}(\boldsymbol{u}), \tag{1}$$

in a three-dimensional domain $\Omega$, with solution $\boldsymbol{u}$, flux function $\boldsymbol{F}(\boldsymbol{u})$, source function $\boldsymbol{S}(\boldsymbol{u})$, and appropriate boundary conditions on $\partial\Omega$. We will use a discretization of $\Omega$ into non-overlapping, conforming, curved hexahedral elements. Within each element we introduce a Cartesian grid of $(p+1)^3$ node points, where $p \geq 1$, by defining a smooth one-to-one mapping given by a diffeomorphism $\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{X})$ between the reference unit cube $V = [0,1]^3$ and the element $v$, and setting $\boldsymbol{x}_{ijk} = \boldsymbol{x}(\boldsymbol{X}_{ijk})$, where $\boldsymbol{X}_{ijk} = (s_i, s_j, s_k)$ for $0 \leq i,j,k \leq p$, and $\{s_i\}$ is an increasing sequence of $p+1$ node positions $s_i \in [0,1]$ with $s_0 = 0$ and $s_p = 1$ (see figure 1).

To obtain our numerical scheme for approximating (1), we consider a single element $v$ and its mapping $\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{X})$, and follow standard procedure to change independent variables from $\boldsymbol{x}$ to $\boldsymbol{X}$. This transforms (1) into

$$J\frac{\partial \boldsymbol{u}}{\partial t} + \nabla_{\boldsymbol{X}} \cdot \widetilde{\boldsymbol{F}}(\boldsymbol{u}) = J\boldsymbol{S}(\boldsymbol{u}), \tag{2}$$

in the reference domain $V$. Here we have defined the mapping Jacobian $J = \det(\boldsymbol{G})$ and the contravariant fluxes $\widetilde{\boldsymbol{F}} = (\widetilde{\boldsymbol{f}}_1, \widetilde{\boldsymbol{f}}_2, \widetilde{\boldsymbol{f}}_3) = J\boldsymbol{G}^{-1}\boldsymbol{F}$, with the mapping deformation gradient $\boldsymbol{G} = \nabla_{\boldsymbol{X}}\boldsymbol{x}$.

A standard nodal discontinuous Galerkin method would now consider the multivariate polynomial $\boldsymbol{u}(\boldsymbol{X})$ that interpolates the grid function, $\boldsymbol{u}_{ijk} = \boldsymbol{u}(\boldsymbol{X}_{ijk})$, and define a numerical scheme for the spatial derivatives of (1) by a Galerkin procedure in $V$. Our approach differs in that it considers each of the three spatial derivatives in (2) separately and approximates them numerically using one-dimensional discontinuous Galerkin formulations along each of the $3(p+1)^2$ curves defined by straight lines in the reference domain $V$, through the three sets of nodes along each space dimension.

More specifically, the $(p+1)^2$ curves along the first space dimension are $\boldsymbol{x}_{jk}(\xi) = \boldsymbol{x}(\xi, X_j, X_k)$ for $0 \leq j,k \leq p$. On these we define the polynomial $\boldsymbol{u}_{jk}(\xi) \in \mathcal{P}_p([0,1])^m$ that interpolates $\boldsymbol{u}_{ijk}$, $i = 0,\ldots,p$, and we define a numerical approximation $\boldsymbol{r}_{jk}(X_1)$ to $\partial\widetilde{\boldsymbol{f}_1}/\partial X_1$ by a one-dimensional Galerkin procedure: Find $\boldsymbol{r}_{jk}(\xi) \in \mathcal{P}_p([0,1])^m$ such that

$$\begin{aligned}\int_0^1 \boldsymbol{r}_{jk}(\xi)\cdot\boldsymbol{v}(\xi)\,d\xi &= \int_0^1 \frac{d\widetilde{\boldsymbol{f}_1}(\boldsymbol{u}_{jk}(\xi))}{d\xi}\cdot\boldsymbol{v}(\xi)\,d\xi \\ &= \widehat{\widetilde{\boldsymbol{f}_1}}(\boldsymbol{u}^+_{jk}(1), \boldsymbol{u}_{jk}(1))\cdot\boldsymbol{v}(1) - \widehat{\widetilde{\boldsymbol{f}_1}}(\boldsymbol{u}_{jk}(0), \boldsymbol{u}^-_{jk}(0))\cdot\boldsymbol{v}(0) - \int_0^1 \widetilde{\boldsymbol{f}_1}(\boldsymbol{u}_{jk}(\xi))\cdot\frac{d\boldsymbol{v}}{d\xi}\,d\xi,\end{aligned} \tag{3}$$

for all test functions $\boldsymbol{v}(\xi) \in \mathcal{P}_p([0,1])^m$. Here, $\boldsymbol{u}^+_{jk}(1)$ is the numerical solution at $\boldsymbol{x}_{jk}(1^+) = \boldsymbol{x}(1^+, X_j, X_k)$, and similarly $\boldsymbol{u}^-_{jk}(0)$ at $\boldsymbol{x}_{jk}(0^-) = \boldsymbol{x}(0^-, X_j, X_k)$. These will be given either by nodes in the neighboring elements or implicitly through the boundary conditions. Furthermore, $\widehat{\widetilde{\boldsymbol{f}_1}}(\boldsymbol{u}_R, \boldsymbol{u}_L)$ is a numerical flux function for $\widetilde{\boldsymbol{f}_1}$, but we note that with the reference normal direction $\boldsymbol{N}_1^+ = (1,0,0)$, the contravariant flux can be written

$$\widetilde{\boldsymbol{f}_1} = \widetilde{\boldsymbol{F}}\cdot\boldsymbol{N}_1^+ = (J\boldsymbol{G}^{-1}\boldsymbol{F})\cdot\boldsymbol{N}_1^+ = \boldsymbol{F}\cdot(J\boldsymbol{G}^{-T}\boldsymbol{N}_1^+) = \boldsymbol{F}\cdot\boldsymbol{n}_1^+ \tag{4}$$

with the (non-normalized) normal vector $\boldsymbol{n}_1^+ = J\boldsymbol{G}^{-T}\boldsymbol{N}_1^+$ at the boundary point $\boldsymbol{x}_{jk}(1)$. Our numerical flux then becomes

$$\widehat{\widetilde{\boldsymbol{f}_1}}(\boldsymbol{u}_R, \boldsymbol{u}_L) = \widehat{\widetilde{\boldsymbol{F}}\cdot\boldsymbol{N}_1^+}(\boldsymbol{u}_R, \boldsymbol{u}_L) = \widehat{\boldsymbol{F}\cdot\boldsymbol{n}_1^+}(\boldsymbol{u}_R, \boldsymbol{u}_L), \tag{5}$$

where $\widehat{\boldsymbol{F}\cdot\boldsymbol{n}}(\boldsymbol{u}^+, \boldsymbol{u}^-)$ is a standard numerical flux function used in finite volume and discontinuous Galerkin schemes, with normal direction $\boldsymbol{n}$ and traces $\boldsymbol{u}^\pm$ in the positive/negative normal direction. This allows us to use existing flux functions and approximate Riemann solvers without modification.

Similarly, for the second numerical flux we move the negative sign to the normal direction and define $\boldsymbol{N}_1^- = (-1,0,0)$ and $\boldsymbol{n}_1^- = J\boldsymbol{G}^{-T}\boldsymbol{N}_1^-$, which is again an outward normal vector. We can then write:

$$-\widehat{\widetilde{\boldsymbol{f}_1}}(\boldsymbol{u}_R, \boldsymbol{u}_L) = \widehat{\widetilde{\boldsymbol{F}}\cdot\boldsymbol{N}_1^-}(\boldsymbol{u}_L, \boldsymbol{u}_R) = \widehat{\boldsymbol{F}\cdot\boldsymbol{n}_1^-}(\boldsymbol{u}_L, \boldsymbol{u}_R), \tag{6}$$

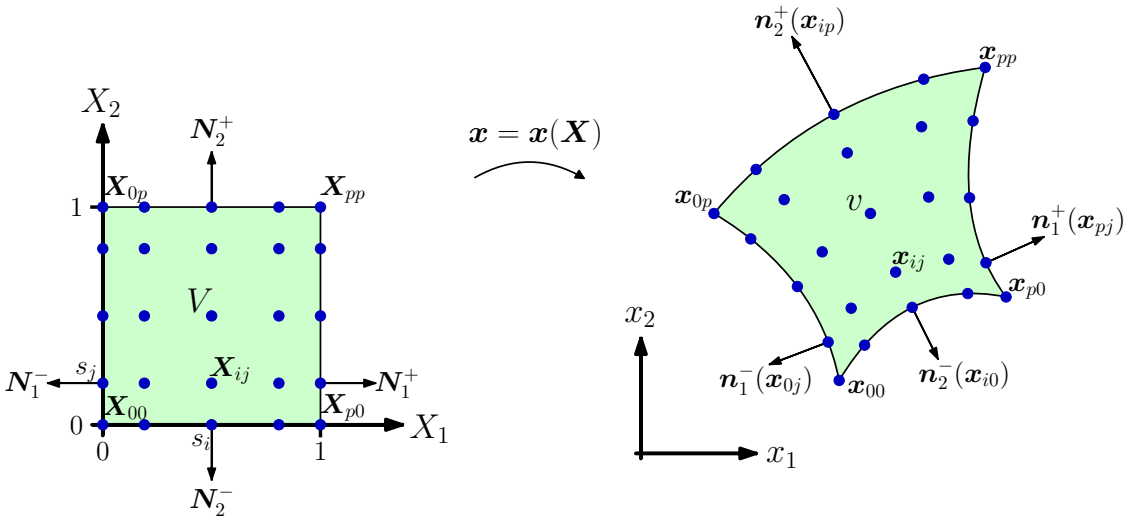



Figure 1: A two-dimensional illustration of the mapping from a reference element $V$ to the actual curved element $v$, for the case $p = 4$.

where we also have swapped the order of the arguments to the flux function $\widehat{\boldsymbol{F}\cdot\boldsymbol{n}}$ to be consistent with the negative normal direction. Our Galerkin scheme (3) then gets the final form: Find $\boldsymbol{r}_{jk}(\xi) \in \mathcal{P}_p([0,1])^m$ such that

$$\int_0^1 \boldsymbol{r}_{jk}(\xi)\cdot\boldsymbol{v}(\xi)\,d\xi = \widehat{\boldsymbol{F}\cdot\boldsymbol{n}_1^+}(\boldsymbol{u}_{jk}^+(1),\boldsymbol{u}_{jk}(1))\cdot\boldsymbol{v}(1) + \widehat{\boldsymbol{F}\cdot\boldsymbol{n}_1^-}(\boldsymbol{u}_{jk}^-(0),\boldsymbol{u}_{jk}(0))\cdot\boldsymbol{v}(0) - \int_0^1 \widetilde{\boldsymbol{f}_1}(\boldsymbol{u}_{jk}(\xi))\cdot\frac{d\boldsymbol{v}}{d\xi}\,d\xi, \tag{7}$$

for all $\boldsymbol{v}(\xi) \in \mathcal{P}_p([0,1])^m$.

We use a standard finite element procedure to solve (7) for $\boldsymbol{r}_{jk}(\xi)$. Introduce the nodal Lagrange basis functions $\phi_i \in \mathcal{P}_p([0,1])$ such that $\phi_i(s_j) = \delta_{ij}$, for $i,j = 0,\ldots,p$, and set

$$\boldsymbol{u}_{jk}(\xi) = \sum_{i=0}^{p} \boldsymbol{u}_{ijk}\phi_i(\xi), \tag{8}$$

$$\boldsymbol{r}_{jk}(\xi) = \sum_{i=0}^{p} \boldsymbol{r}_{ijk}\phi_i(\xi), \tag{9}$$

To find the $m(p+1)$ coefficients along the curve $\boldsymbol{x}_{jk}(\xi)$, we set $\boldsymbol{v}(\xi) = \boldsymbol{e}_\ell\phi_i(\xi)$, for each $i = 0,\ldots,p$ and $\ell = 1,\ldots,m$, where $(\boldsymbol{e}_\ell)_n = \delta_{\ell n}$ for $n = 1,\ldots,m$. Our Galerkin scheme (7) then gets the discrete form $\boldsymbol{M}\boldsymbol{r}_{jk} = \boldsymbol{b}$, and we find the coefficients $\boldsymbol{r}_{jk}$ by solving $m$ linear systems with the $(p+1)$-by-$(p+1)$ mass matrix $\boldsymbol{M}$. Repeating the procedure for each $j,k = 0,\ldots,p$ we obtain all coefficients $\boldsymbol{r}_{ijk} = \boldsymbol{r}_{ijk}^{(1)}$, which is the grid function for our numerical approximation of $\partial\widetilde{\boldsymbol{F}}_1/\partial X_1$ at each grid point $\boldsymbol{x}_{ijk}$.

In an analogous way, we calculate coefficients $\boldsymbol{r}_{ijk}^{(2)}$ and $\boldsymbol{r}_{ijk}^{(3)}$ that approximate $\partial\widetilde{\boldsymbol{F}}_2/\partial X_2$ and $\partial\widetilde{\boldsymbol{F}}_3/\partial X_3$, respectively, at the grid points. The curves considered are now $\boldsymbol{x}_{ik} = \boldsymbol{x}(X_i,\xi,X_k)$ and $\boldsymbol{x}_{ij} = \boldsymbol{x}(X_i,X_j,\xi)$, and with the reference normals $\boldsymbol{N}_2^\pm = (0,\pm1,0)$ and $\boldsymbol{N}_3^\pm = (0,0,\pm1)$ the contravariant fluxes $\widetilde{\boldsymbol{f}}_2$ and $\widetilde{\boldsymbol{f}}_3$ can again be written as $\boldsymbol{F}\cdot\boldsymbol{n}$ where $\boldsymbol{n} = J\boldsymbol{G}^{-T}\boldsymbol{N}$ is a non-normalized normal vector to the element at the boundary points. The solution procedure involves the same mass matrix $\boldsymbol{M}$ and is identical to before.

Using the calculated numerical approximations to each partial derivative in (2), we obtain our final

semi-discrete formulation:

$$\frac{d\boldsymbol{u}_{ijk}}{dt} + \frac{1}{J_{ijk}} \sum_{n=1}^{3} \boldsymbol{r}_{ijk}^{(n)} = \boldsymbol{S}(\boldsymbol{u}_{ijk}), \tag{10}$$

where $J_{ijk} = J(\boldsymbol{x}_{ijk})$.

### *2.2. Implementation details*

For the mapping $\boldsymbol{x}(\boldsymbol{X})$ it is natural to use an iso-parametric approach. The node positions $\boldsymbol{x}_{ijk}$ are given by some curved mesh generation procedure [21], and we define

$$\boldsymbol{x}(\boldsymbol{X}) = \sum_{i,j,k=0}^{p} \boldsymbol{x}_{ijk} \phi_i(X_1) \phi_j(X_2) \phi_k(X_3), \tag{11}$$

which clearly satisfies our interpolation requirement

$$\boldsymbol{x}(\boldsymbol{X}_{ijk}) = \sum_{i',j',k'=0}^{p} \boldsymbol{x}_{i'j'k'} \phi_{i'}(s_i) \phi_{j'}(s_j) \phi_{k'}(s_k) = \sum_{i',j',k'=0}^{p} \boldsymbol{x}_{i'j'k'} \delta_{ii'} \delta_{jj'} \delta_{kk'} = \boldsymbol{x}_{ijk}. \tag{12}$$

This allows us to easily compute $\boldsymbol{G}(\boldsymbol{X})$ at any point $\boldsymbol{X}$, which will involve the derivatives $\phi_i'(\xi)$ of the shape functions. To evaluate the one-dimensional integrals in (7), we use Gauss-Legendre integration of sufficiently high degree. For all our problems, a precision of $3p$ appears to be enough, so we use integration rules with $\lceil (3p+1)/2 \rceil$ integration points.

The computation of the discretization (7) is remarkably simple compared to a nodal DG scheme, primarily because (a) The integrals are only one-dimensional, and (b) The numerical fluxes are only evaluated point-wise. We note that the left-hand side of (7), which contributes to the mass matrix $\boldsymbol{M}$, is constant regardless of solution component, line, and element (even if the actual mapped element is curved). Therefore it can be pre-computed and pre-factorized using a standard Cholesky method. Furthermore, many lines and components can be processed simultaneously, which might further increase the performance through the use of BLAS3-type cache-optimized linear algebra libraries.

For the integral in the right-hand side of (7), the term $d\boldsymbol{v}/d\xi$ is again constant for all components, lines, and elements, so its discretization at the Gauss integration points can be pre-computed and combined with the inverted mass matrix and the Gauss integration weights $\boldsymbol{w}$. For non-linear problems, the only part that requires re-evaluation at each Gauss integration point is $\widetilde{\boldsymbol{f}}_1(\boldsymbol{u}_{jk}(\xi))$, although the deformation gradient $\boldsymbol{G}$ can be pre-computed if necessary.

For the numerical fluxes, we pointed out above that (5) and (6) have exactly the same form as standard numerical flux functions. We pre-compute the outward normals $\boldsymbol{n}_i^+$ and $\boldsymbol{n}_i^-$, for $i = 1, 2, 3$, at all boundary nodes. Note that our scheme only computes point-wise numerical fluxes, unlike the nodal DG method which involves integrals of the numerical fluxes. Sometimes existing numerical flux functions require the normal vector to be of unit length, in this case we normalize $\bar{\boldsymbol{n}} = \boldsymbol{n}/|\boldsymbol{n}|$ and use the fact that

$$\widehat{\boldsymbol{F} \cdot \boldsymbol{n}} = |\boldsymbol{n}| \widehat{\boldsymbol{F} \cdot \bar{\boldsymbol{n}}}. \tag{13}$$

Finally, the multipliers $J_{ijk}$ are defined at the node points (not the Gauss integration points), and can also be pre-computed.

### *2.3. Stencil size and sparsity pattern*

To illustrate the drastic reduction of the number of entries in the Jacobian matrices for the Line-DG method, consider the $(p+1)^3$ nodes in an (interior) element and its six neighboring elements. For a first-order operator, we note that a standard nodal DG formulation will in general produce full block matrices, that is, each degree of freedom will depend on all the other ones within the element. In addition, the

| | Polynomial order $p$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2-D | Line-DG connectivities | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 |
| | DGSEM/SD connectivities | 11 | 17 | 23 | 29 | 35 | 41 | 47 | 53 | 59 | 65 |
| | Nodal DG connectivities | 8 | 13 | 20 | 29 | 40 | 53 | 68 | 85 | 104 | 125 |
| 3-D | Line-DG connectivities | 10 | 13 | 16 | 19 | 22 | 25 | 28 | 31 | 34 | 37 |
| | DGSEM/SD connectivities | 16 | 25 | 34 | 43 | 52 | 61 | 70 | 79 | 88 | 97 |
| | Nodal DG connectivities | 20 | 45 | 88 | 155 | 252 | 385 | 560 | 783 | 1060 | 1397 |

Table 1: The number of connectivities per node for a first-order operator and 2-D quadrilateral / 3-D hexahedral elements with the Line-DG, the DGSEM/SD, and the nodal DG methods.

face integrals will connect all nodes on an element face to all neighboring element face nodes. This gives $6(p+1)^2(p+1)^2 = 6(p+1)^4$ additional connections per element, or in average $6(p+1)^4/(p+1)^3 = 6(p+1)$ connections per degree of freedom. In total, the average number of connections is $(p+1)^3 + 6(p+1)$, which illustrates why matrix-based DG methods are considered memory intensive and expensive even at modest values of $p$.

As a contrast, in our line-based method each node will only connect to other nodes within the same lines, and to only one node in each neighboring element, for a total of $(3p+1)+6 = 3p+7$ connectivities. This is similar to that of the DGSEM/SD methods, although with Gauss-Legendre solution points these schemes also connect entire lines of nodes in the neighboring element, giving a total of $(3p+1)+6(p+1) = 9p+7$ connectivities. These numbers are tabulated for a range of degrees $p$ in three dimensions in table 1. The sparsity patterns are illustrated in figure 2 for two-dimensional quadrilateral elements, for all three methods. The connectivities are shown both by a nodal plot, with bold nodes corresponding to the dependencies of the single red node, and by sparsity plots of the Jacobian matrices.

We note that in three dimensions, already for $p = 3$ the Line-DG method is 5.5 times sparser than nodal DG, and for $p = 10$ it is almost 40 times sparser. This reduction in stencil size translates into lower assembly times, but more importantly, for matrix-based solvers it means drastically lower storage requirements and faster matrix-vector products for iterative implicit solvers.

## 3. Second-order equations

We now consider the discretization of equations with second-order derivatives, in the form of a system of conservation laws

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}, \nabla \boldsymbol{u}) = \boldsymbol{S}(\boldsymbol{u}, \nabla \boldsymbol{u}). \tag{14}$$

We first use a standard technique in many finite difference and discontinuous Galerkin methods, and introduce the auxiliary variables $\boldsymbol{q}$ and rewrite as a split system

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q}) = \boldsymbol{S}(\boldsymbol{u}, \boldsymbol{q}), \tag{15}$$

$$\nabla \boldsymbol{u} = \boldsymbol{q}. \tag{16}$$

This essentially has the form of our first-order system (1), and we can apply Line-DG to each solution component as described above. More specifically, the change of variables from $\boldsymbol{x}$ to $\boldsymbol{X}$ transforms (15), (16) into

$$J\frac{\partial \boldsymbol{u}}{\partial t} + \nabla_{\boldsymbol{X}} \cdot \widetilde{\boldsymbol{F}}(\boldsymbol{u}, \boldsymbol{q}) = J\boldsymbol{S}(\boldsymbol{u}, \boldsymbol{q}), \tag{17}$$

$$\nabla_{\boldsymbol{X}} \cdot \widetilde{\boldsymbol{u}}(\boldsymbol{u}) = J\boldsymbol{q}, \tag{18}$$

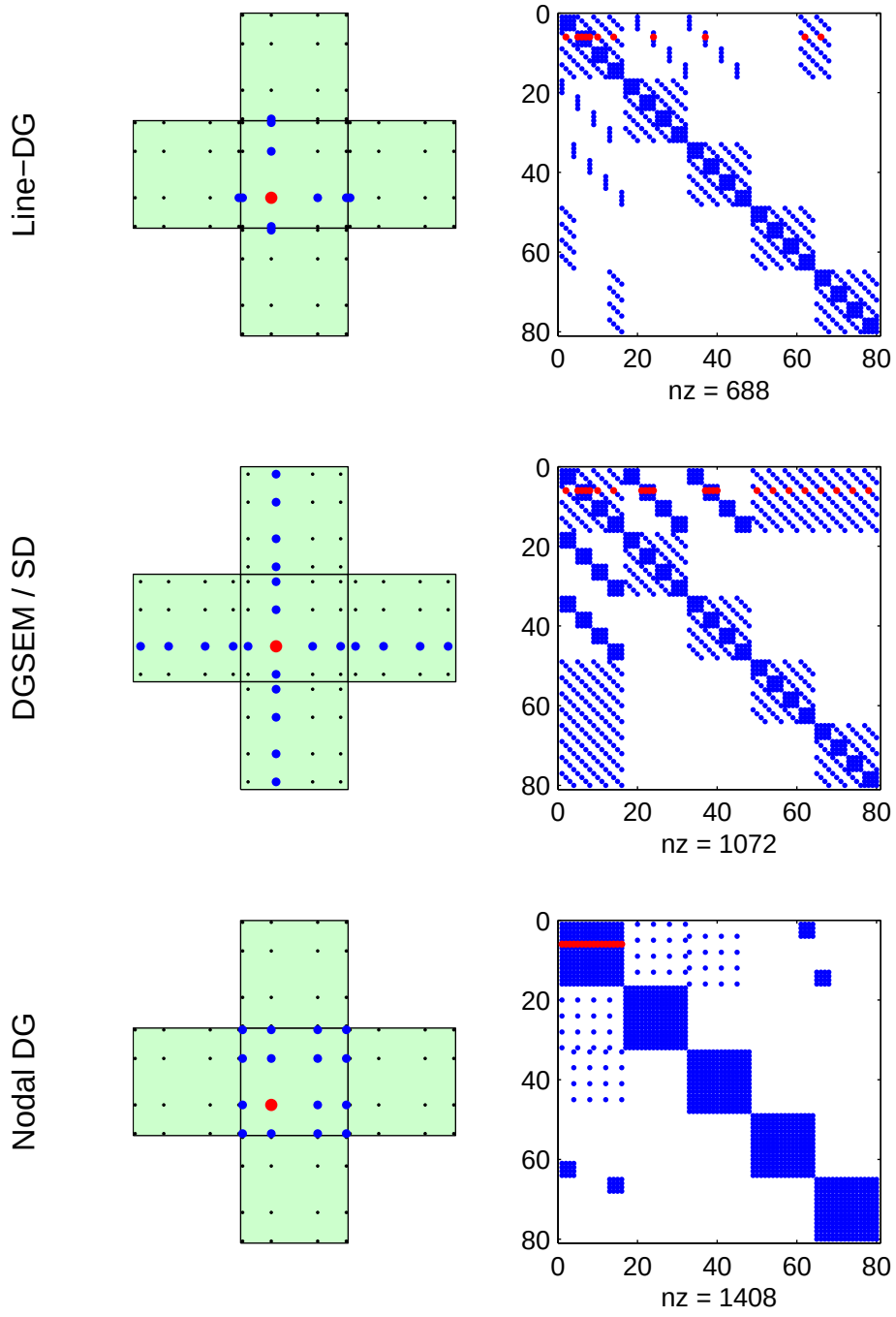



Figure 2: The connectivities (blue circles) to a single node (red circle) for the Line-DG method, the DGSEM/SD method, and the nodal DG method (2-D quadrilateral elements, a first-order operator).

where $\widetilde{\boldsymbol{u}} = (\widetilde{\boldsymbol{u}}_1, \widetilde{\boldsymbol{u}}_2, \widetilde{\boldsymbol{u}}_3) = \boldsymbol{u} \otimes J\boldsymbol{G}^{-1}$. We discretize (17) as described before for the first-order case, treating $\boldsymbol{q}$ as additional solution components. For (18), we use a completely analogous procedure. We introduce the grid function $\boldsymbol{q}_{ijk} = \boldsymbol{q}(\boldsymbol{X}_{ijk})$, and along each curve $\boldsymbol{X}_{jk}(\xi)$ we define the polynomial $\boldsymbol{q}_{jk}(\xi) \in \mathcal{P}_p([0,1])^{m\times 3}$ that interpolates $\boldsymbol{q}_{ijk}$, $i = 0, \ldots, p$. We find the numerical approximation $\boldsymbol{d}_{jk}(X_1)$ to $\partial\widetilde{\boldsymbol{u}}_1/\partial X_1$ by the Galerkin formulation: Find $\boldsymbol{d}_{jk}(\xi) \in \mathcal{P}_p([0,1])^{m\times 3}$ such that

$$\begin{aligned}\int_0^1 \boldsymbol{d}_{jk}(\xi) : \boldsymbol{\tau}(\xi)\, d\xi &= \int_0^1 \frac{d\widetilde{\boldsymbol{u}}_1}{d\xi} : \boldsymbol{\tau}(\xi)\, d\xi \\ &= \widehat{\widetilde{\boldsymbol{u}}}_1(\boldsymbol{u}^+_{jk}(1), \boldsymbol{q}^+_{jk}(1), \boldsymbol{u}_{jk}(1), \boldsymbol{q}_{jk}(1)) : \boldsymbol{\tau}(1) - \widehat{\widetilde{\boldsymbol{u}}}_1(\boldsymbol{u}_{jk}(0), \boldsymbol{q}_{jk}(0), \boldsymbol{u}^-_{jk}(0), \boldsymbol{q}^-_{jk}(0)) : \boldsymbol{\tau}(0) \\ &\quad - \int_0^1 \widetilde{\boldsymbol{u}}_1(\boldsymbol{u}_{jk}(\xi)) : \frac{d\boldsymbol{\tau}}{d\xi}\, d\xi \end{aligned} \tag{19}$$

for all test functions $\boldsymbol{\tau}(\xi) \in \mathcal{P}_p([0,1])^{m\times 3}$. Note that we allow for the numerical flux $\widehat{\widetilde{\boldsymbol{u}}}_1$ to depend on both $\boldsymbol{u}$ and $\boldsymbol{q}$ on each side of the face, even though the actual flux $\widetilde{\boldsymbol{u}}_1$ is only a function of $\boldsymbol{u}$. Again, the numerical contravariant fluxes can be written in terms of the actual fluxes and the actual normal vector:

$$\widehat{\widetilde{\boldsymbol{u}}_1} = \widehat{\widetilde{\boldsymbol{u}} \cdot \boldsymbol{N}_1^+} = \widehat{\boldsymbol{u} \otimes \boldsymbol{n}_1^+} = \widehat{\boldsymbol{u}} \otimes \boldsymbol{n}_1^+, \tag{20}$$

and similarly in the negative direction and along the other coordinate directions. It remains only to define the numerical fluxes $\widehat{\boldsymbol{F} \cdot \boldsymbol{n}} = \widehat{\boldsymbol{F}} \cdot \boldsymbol{n}$ and $\widehat{\boldsymbol{u}}$. We could in principle consider any scheme that can be written in this form [22], such as the interior penalty method, the BR2 method, the LDG method [20], and the CDG method [23]. Here we use a scheme based on the LDG method, because it has a simple upwind/downwind character, it does not evaluate derivatives of grid functions at the boundaries, and it appears well-suited for our Line-DG discretization. Furthermore, since our implicit solvers avoid the elimination of $\boldsymbol{q}$, the scheme has a compact connectivity (only connects neighboring elements).

First, we separate the fluxes $\boldsymbol{F}$ into an inviscid and a viscous part:

$$\boldsymbol{F}(\boldsymbol{u}, \nabla\boldsymbol{u}) = \boldsymbol{F}^{\text{inv}}(\boldsymbol{u}) + \boldsymbol{F}^{\text{vis}}(\boldsymbol{u}, \nabla\boldsymbol{u}). \tag{21}$$

This decomposition is clearly not unique, but it is understood that for many problems there is a natural separation into a convection-dominated inviscid component and a diffusion-dominated viscous component. This allows us to use standard approximate Riemann solvers for $\widehat{\boldsymbol{F}}^{\text{inv}}$ as before, and we will now consider only the treatment of the viscous fluxes $\widehat{\boldsymbol{F}}^{\text{vis}}$. For shorter notation, we assume below that $\boldsymbol{F} = \boldsymbol{F}^{\text{vis}}$ and that $\boldsymbol{n}$ is a unit vector.

We will define the fluxes in terms of a so-called switch function, which simply assigns a sign to each internal element face. For one-dimensional problems, the natural switch function is to set all these signs equal (either positive or negative), and we will mimic this for our Line-DG method by identifying globally connected lines in our hexahedral meshes.

In our notation, instead of assigning switches to each face, we introduce $S_i^{\pm} \in \{-1, 1\}$ for the switches at local coordinate $\xi = 1$ and $\xi = 0$ along direction $i = 1, 2, 3$. There is some redundancy here, since we require that $S_i^+ = -S_i^-$, and also that the switch function for a shared face between two neighboring elements have opposite signs. See figure 3 for an example quadrilateral mesh and switch function. This was generated by a straight-forward algorithm, where an arbitrary element face is chosen and assigned an arbitrary sign, which then defines the alternating pattern along a sequence of elements in both directions. This procedure is repeated until all faces have been processed.

With the switch function defined, we can formulate the LDG numerical fluxes for the second-order terms:

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{n}) = \{\{\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q})\}\} + C_{11}[\![\boldsymbol{u} \otimes \boldsymbol{n}]\!] + \boldsymbol{C}_{12} \otimes [\![\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q}) \cdot \boldsymbol{n}]\!], \tag{22}$$

$$\widehat{\boldsymbol{u}}(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{n}) = \{\{\boldsymbol{u}\}\} - \boldsymbol{C}_{12} \cdot [\![\boldsymbol{u} \otimes \boldsymbol{n}]\!] + C_{22}[\![\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q}) \cdot \boldsymbol{n}]\!]. \tag{23}$$

for a solution $\boldsymbol{u}, \boldsymbol{q}$ and a face normal vector $\boldsymbol{n}$. Here, $\{\{\cdot\}\}$ denotes the mean value and $[\![\cdot]\!]$ denotes the jump over a face:

$$\{\{\boldsymbol{v}\}\} \equiv \frac{1}{2}(\boldsymbol{v}^+ + \boldsymbol{v}^-), \qquad [\![\boldsymbol{v} \odot \boldsymbol{n}]\!] \equiv \boldsymbol{v}^+ \odot \boldsymbol{n} - \boldsymbol{v}^- \odot \boldsymbol{n} \tag{24}$$

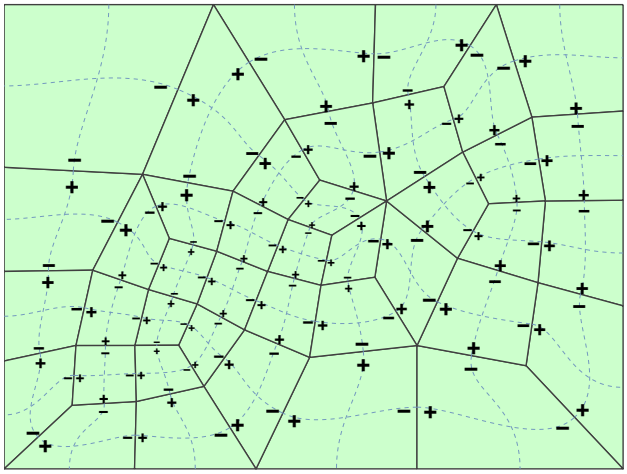


Figure 3: A sample quadrilateral mesh and switch function $S_i^{\pm}$ for $i = 1, 2$ in each element. Note that the switches have consistent directions along each one-dimensional global curve (dashed blue lines).

where $\boldsymbol{v}^+$ is the quantity $\boldsymbol{v}$ on the positive side of the face (according to the normal $\boldsymbol{n}$), $\boldsymbol{v}^-$ is $\boldsymbol{v}$ on the negative side, and $\odot$ is any multiplication operator. The coefficients $C_{11}, \boldsymbol{C}_{12}, C_{22}$ give the scheme different properties, and we note in particular that:

- If $C_{22} = 0$, the fluxes (23) do not depend on $\boldsymbol{q}$, which means the discretized equation (18) immediately give $\boldsymbol{q}$ within each element (no coupling to equation (17)).
- For the particular choice $\boldsymbol{C}_{12} = \boldsymbol{n} S_i^{\pm}/2$, where $S_i^{\pm}$ is the switch for the considered face and direction, the fluxes can be written in the form:

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}_R, \boldsymbol{q}_R, \boldsymbol{u}_L, \boldsymbol{q}_L, \boldsymbol{n}) = C_{11}[\![\boldsymbol{u} \otimes \boldsymbol{n}]\!] + \begin{cases} \boldsymbol{F}(\boldsymbol{u}_R, \boldsymbol{q}_R) & \text{if } S_i^{\pm} = +1 \\ \boldsymbol{F}(\boldsymbol{u}_L, \boldsymbol{q}_L) & \text{if } S_i^{\pm} = -1 \end{cases} \tag{25}$$

$$\widehat{\boldsymbol{u}}(\boldsymbol{u}_R, \boldsymbol{q}_R, \boldsymbol{u}_L, \boldsymbol{q}_L, \boldsymbol{n}) = C_{22}[\![\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q}) \cdot \boldsymbol{n}]\!] + \begin{cases} \boldsymbol{u}_L & \text{if } S_i^{\pm} = +1 \\ \boldsymbol{u}_R & \text{if } S_i^{\pm} = -1 \end{cases} \tag{26}$$

  where it is clear how the method is upwinding/downwinding the two numerical fluxes, depending on the switch $S_i^{\pm}$. The constants $C_{11}$ and $C_{22}$ are additional stabilization parameters, which can be seen as penalties on the jumps in the solution and in the normal fluxes, respectively. In many of our problems we set both of these coefficients to zero (the so-called minimal dissipation LDG method [24]). This makes the scheme particularly simple, and also further reduces the number of connectivities in the Jacobian matrices.

At the boundaries we impose conditions by appropriate choices of numerical fluxes. For example, at a Dirichlet-type boundary with a prescribed solution $\boldsymbol{u} = \boldsymbol{g}_D$, we set:

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{n}) = \boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q}) + C_{11}(\boldsymbol{u} - \boldsymbol{g}_D) \otimes \boldsymbol{n} \tag{27}$$

$$\widehat{\boldsymbol{u}}(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{n}) = \boldsymbol{g}_D, \tag{28}$$

where $C_{11}$ in (27) must be positive, even though we often choose $C_{11} = 0$ for the interior fluxes. At a Neumann-type boundary with prescribed normal fluxes $\boldsymbol{F} \cdot \boldsymbol{n} = \boldsymbol{g}_N$, we set:

$$\widehat{\boldsymbol{F}}(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{n}) = \boldsymbol{g}_N \otimes \boldsymbol{n} \tag{29}$$

$$\widehat{\boldsymbol{u}}(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{n}) = \boldsymbol{u} - C_{22}(\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{q}) \cdot \boldsymbol{n} - \boldsymbol{g}_N). \tag{30}$$

For mixed conditions we apply combinations of these fluxes for the different components of $\boldsymbol{u}$ and $\boldsymbol{q}$.

With the fluxes defined, we can calculate $\boldsymbol{d}_{ijk} = \boldsymbol{d}^{(1)}_{ijk}$ for all $j, k = 0, \ldots, p$, and similarly for $\boldsymbol{d}^{(2)}_{ijk}$ and $\boldsymbol{d}^{(3)}_{ijk}$ along the other two coordinate directions. These are essentially numerical approximations to the gradient $\boldsymbol{q} = \nabla \boldsymbol{u}$, but again we point out that they might depend implicitly on $\boldsymbol{q}$ through the numerical fluxes (if $C_{22} \neq 0$). Our final semi-discrete formulation for (17), (18) gets the form

$$\frac{d\boldsymbol{u}_{ijk}}{dt} + \frac{1}{J_{ijk}} \sum_{n=1}^{3} \boldsymbol{r}^{(n)}_{ijk} = \boldsymbol{S}(\boldsymbol{u}_{ijk}, \boldsymbol{q}_{ijk}) \tag{31}$$

$$\frac{1}{J_{ijk}} \sum_{n=1}^{3} \boldsymbol{d}^{(n)}_{ijk} = \boldsymbol{q}_{ijk} \tag{32}$$

## 4. Temporal discretization and nonlinear solvers

### *4.1. Method of lines and time integration*

We use various techniques to solve the semi-discrete system of equations (31), (32), either by integrating in time or solving for steady-state solutions. First, we define the vectors $\boldsymbol{U}, \boldsymbol{Q}$ with all solution components $\boldsymbol{u}_{ijk}, \boldsymbol{q}_{ijk}$, respectively, and write the system as

$$\frac{d\boldsymbol{U}}{dt} = \boldsymbol{R}(\boldsymbol{U}, \boldsymbol{Q}), \tag{33}$$

$$\boldsymbol{Q} = \boldsymbol{D}(\boldsymbol{U}, \boldsymbol{Q}). \tag{34}$$

This split form can be useful for implicit time-stepping or steady-state solutions, in particular if the coefficient $C_{22} \neq 0$. With a standard Newton's method, this requires the solution of linear systems involving the matrix

$$\boldsymbol{K} = \begin{bmatrix} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}} & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}} \\ \frac{\partial \boldsymbol{D}}{\partial \boldsymbol{U}} & \frac{\partial \boldsymbol{D}}{\partial \boldsymbol{Q}} \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{12} \\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22} \end{bmatrix}. \tag{35}$$

This system solves for both $\boldsymbol{U}$ and $\boldsymbol{Q}$ but it retains the high level of sparsity of the method. Also, it allows for non-zero $\boldsymbol{K}_{22}$ which can be used to give the scheme several attractive properties [15]. In our examples, we solve these equations using a standard sparse direct solver [25].

However, in most of our problems we set $C_{22} = 0$ to allow for elimination of the discrete derivatives $\boldsymbol{Q}$. Then $\boldsymbol{D}(\boldsymbol{U}, \boldsymbol{Q}) = \boldsymbol{D}(\boldsymbol{U})$, and substituting (34) into (33) leads to a reduced system

$$\frac{d\boldsymbol{U}}{dt} = \boldsymbol{R}(\boldsymbol{U}, \boldsymbol{D}(\boldsymbol{U})) \equiv \boldsymbol{F}(\boldsymbol{U}). \tag{36}$$

This is clearly the preferred choice for explicit time-stepping, since it is a regular system of ODEs. In our examples we use a standard fourth-order explicit Runge-Kutta method. We also use this form for implicit time-stepping using Diagonally Implicit Runge-Kutta (DIRK) schemes [26]. In particular, we use the following L-stable, three-stage, third-order accurate method [26]:

$$\boldsymbol{K}_i = \boldsymbol{F}\left(\boldsymbol{U}_n + \Delta t \sum_{j=1}^{s} a_{ij} \boldsymbol{K}_j\right), \quad i = 1, \ldots, s \tag{37}$$

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \Delta t \sum_{j=1}^{s} b_j \boldsymbol{K}_j, \tag{38}$$

with $s = 3$ and the coefficients given by the Runge-Kutta tableaux below.

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|ccc} \alpha & \alpha & 0 & 0 \\ \tau_2 & \tau_2 - \alpha & \alpha & 0 \\ 1 & b_1 & b_2 & \alpha \\ \hline & b_1 & b_2 & \alpha \end{array} \qquad \begin{aligned} \alpha &= 0.435866521508459 \\ \tau_2 &= (1+\alpha)/2 \\ b_1 &= -(6\alpha^2 - 16\alpha + 1)/4 \\ b_2 &= (6\alpha^2 - 20\alpha + 5)/4 \end{aligned}$$

We also use implicit time-stepping for computing steady-state solutions, by a sequence of increasing timesteps $\Delta t$ and a final step without the time derivatives. Since this does not require time-accuracy, we use a standard backward Euler scheme. We solve the nonlinear systems (37) using Newton's method with preconditioned iterative solvers, as described below.

*4.2. Newton-Krylov solvers*

When Newton's method is applied to the reduced problem (36), it requires the solution of systems of equations of the form

$$(\boldsymbol{I} - \alpha\Delta t \boldsymbol{A})\Delta \boldsymbol{U}^{(i)} = \Delta t \boldsymbol{R}(\boldsymbol{U}^{(i)}, \boldsymbol{D}(\boldsymbol{U}^{(i)})) \tag{39}$$

where $\Delta t$ is the timestep and

$$\boldsymbol{A} = \frac{d\boldsymbol{R}}{d\boldsymbol{U}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}\frac{\partial \boldsymbol{D}}{\partial \boldsymbol{U}} = \boldsymbol{K}_{11} + \boldsymbol{K}_{12}\boldsymbol{K}_{21}. \tag{40}$$

Forming this matrix $\boldsymbol{A}$ has the drawback that for second-order systems, the product $\boldsymbol{K}_{12}\boldsymbol{K}_{21}$ is in general much less sparse than the individual matrices $\boldsymbol{K}_{11}, \boldsymbol{K}_{12}, \boldsymbol{K}_{21}$. This is expected due to the repeated differentiation along two different directions, but it requires special solvers to avoid explicitly forming the denser matrix $\boldsymbol{A}$. This phenomenon is not unique for our method, in fact many other numerical schemes including finite difference methods and nodal DG methods suffer from sparsity reduction for second-order systems.

In this work, we use a simple approach to solve the system (39) without forming the full Jacobian matrix. In a preconditioned Krylov subspace method, we need to perform two operations: Multiplication of a vector $\boldsymbol{p}$ by the matrix $(\boldsymbol{I} - \alpha\Delta t\boldsymbol{A})$, and approximate solution of $(\boldsymbol{I} - \alpha\Delta t\boldsymbol{A})\boldsymbol{x} = \boldsymbol{b}$ for the preconditioning. The matrix-vector product can by computed by keeping the individual matrix in a separated form and nesting the products:

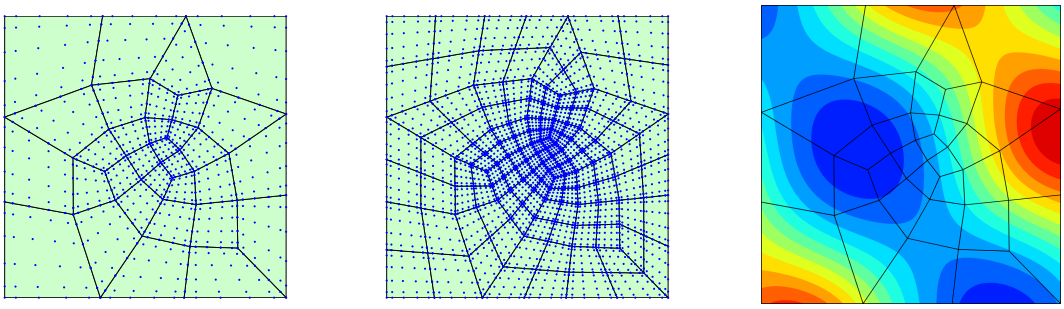
$$(\boldsymbol{I} - \alpha\Delta t\boldsymbol{A})\boldsymbol{p} = \boldsymbol{p} - \alpha\Delta t\left(\boldsymbol{K}_{11}\boldsymbol{p} + \boldsymbol{K}_{12}(\boldsymbol{K}_{21}\boldsymbol{p})\right). \tag{41}$$

This avoids explicitly forming the matrix $\boldsymbol{A}$, and the cost per matrix-vector product is proportional to the number of entries in the matrices $\boldsymbol{K}_{11}, \boldsymbol{K}_{12}, \boldsymbol{K}_{21}$.

For preconditioning, we use a sparse block-Jacobi approach which forms an approximate matrix $\widetilde{\boldsymbol{A}}$ that ignores all fill from the product $\boldsymbol{K}_{12}\boldsymbol{K}_{21}$ and any inter-element connectivities. In other words, $\widetilde{\boldsymbol{A}}$ is equal to $\boldsymbol{A}$ only at the block-diagonal Line-DG sparsity pattern, and zero everywhere else. This simple preconditioner requires very little storage (even less than a first-order discretization or the matrix $\boldsymbol{K}_{11}$) and it performs well for time-accurate simulations with small timesteps. It can certainly be improved upon, for example allowing higher levels of fill or using block-ILU and $h/p$-multigrid schemes [27].

When solving the linear systems involving the preconditioning matrix $(\boldsymbol{I} - \alpha\Delta t\widetilde{\boldsymbol{A}})$, we use a sparse direct LU-factorization with fill-reducing ordering for each block [25]. This results in some additional fill, but in our 2-D examples we find that even for $p$ as high as 7 the number of entries in $\widetilde{\boldsymbol{A}}$ is about the same as in the original sparse matrices $\boldsymbol{K}_{11}$, $\boldsymbol{K}_{12}$, and $\boldsymbol{K}_{21}$. For 3-D problems, it is more critical to retain the line-based sparsity in the preconditioner, and a number of alternatives should be applicable such as low-order approximations [28], ADI-iterations [29], and subiterations [30].

In our implementation, we store all matrices in a general purpose compressed column storage format [25]. We also point out that the matrix $\boldsymbol{K}_{21}$ can be handled very efficiently, since it is a discrete gradient operator and therefore (a) linear, (b) constant in time, and (c) equal for all solution components (except possibly at the boundaries for certain boundary conditions).



Coarsest mesh, $p = 5$ | One refinement, $p = 5$ | Solution $u(x, y)$

Figure 4: The Poisson test problem (42). The left figure shows the coarse unstructured quadrilateral mesh, which is uniformly refined repeatedly, and the solution nodes for $p = 5$. The right figure shows the solution as colored contours.

### *4.3. Re-using Jacobian matrices*

In many problems it is more computationally expensive to form the matrix $\boldsymbol{I} - \alpha\Delta t\boldsymbol{A}$ than to solve the linear system (39). This is especially true for time-accurate integration where the timesteps $\Delta t$ are relatively small (but still large enough to motivate the use of implicit solvers). We therefore use a standard technique for Newton's method that attempts to re-use old Jacobian matrices for many iterations, until the convergence is too slow as determined by the number of iterations exceeding a threshold number. In our numerical experiments this is sufficient to allow for a reuse of the Jacobian for a large number of Newton steps, DIRK stages, and timesteps at a time.

The linear systems are solved using a preconditioned GMRES method [31], with the low-cost sparse block-Jacobi preconditioner $\widetilde{\boldsymbol{A}}$ described above. These equations can be solved with relatively low accuracy using a small number of GMRES iterations, since we are using old Jacobian matrices which already limit the potential improvements from each Newton step. The tolerance in the Newton solver is set well below the estimated truncation error of the time integrator.

## 5. Results

### *5.1. Poisson's equation*

Our first test is Poisson's equation

$$-\nabla \cdot (\nabla u) = f(x, y) \tag{42}$$

on the unit square domain $\Omega = [0, 1]^2$. Dirichlet conditions are imposed at all the boundaries ($\partial\Omega_D = \partial\Omega$) and we choose the analytical solution

$$u(x, y) = \exp\left[\alpha \sin(ax + by) + \beta \cos(cx + dy)\right] \tag{43}$$

with numerical parameters $\alpha = 0.1, \beta = 0.3, a = 5.1, b = -6.2, c = 4.3, d = 3.4$. We then solve (42) with Dirichlet boundary conditions $g_D(x, y) = u(x, y)|_{\partial\Omega_D}$. The source term, $f(x, y)$, is obtained by analytical differentiation of (43).

We discretize $\Omega$ using an unstructured mesh of quadrilateral elements, see Figure 4 (left). We solve the split system (33), (34) using a direct sparse solver, for polynomial degrees $p = 1, \ldots, 7$. We consider two sets of parameters: the minimal dissipation scheme with $C_{11} = C_{22} = 0$, which has the benefit that it allows for elimination of the gradients $\boldsymbol{q}$, and the slightly over-stabilized scheme $C_{11} = C_{22} = 1/20$, which may provide a higher-order of convergence for $\boldsymbol{q}$ [15].

The resulting infinity norm errors and rates of convergence are shown in table 2, for both the solution $u$ and the gradients $\boldsymbol{q}$ and the two parameter cases. For the minimal dissipation scheme (top two tables), we observe the expected orders of convergence $p + 1$ and $p$ for $u$ and $\boldsymbol{q}$, respectively. For the stabilized scheme (bottom two tables), we obtain a somewhat higher order for the $\boldsymbol{q}$ variables, which could be used as part of a postprocessing step to further increase the order of convergence for the solution $u$ [16].

**Error in solution** $|u_h - u|_\infty$, $C_{11} = C_{22} = 0$, $\boldsymbol{C}_{12} = \boldsymbol{n}S_i^{\pm}/2$

| | $p=1$ | | $p=2$ | | $p=3$ | | $p=4$ | | $p=5$ | | $p=6$ | | $p=7$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate |
| 1 | $4.6 \cdot 10^{-2}$ | | $3.7 \cdot 10^{-3}$ | | $4.1 \cdot 10^{-4}$ | | $1.0 \cdot 10^{-4}$ | | $8.9 \cdot 10^{-6}$ | | $2.1 \cdot 10^{-6}$ | | $3.7 \cdot 10^{-7}$ | |
| 2 | $1.1 \cdot 10^{-2}$ | 2.0 | $4.3 \cdot 10^{-4}$ | 3.1 | $2.8 \cdot 10^{-5}$ | 3.9 | $2.5 \cdot 10^{-6}$ | 5.4 | $1.3 \cdot 10^{-7}$ | 6.0 | $1.5 \cdot 10^{-8}$ | 7.1 | $9.4 \cdot 10^{-10}$ | 8.6 |
| 4 | $2.7 \cdot 10^{-3}$ | 2.1 | $5.1 \cdot 10^{-5}$ | 3.1 | $1.7 \cdot 10^{-6}$ | 4.0 | $6.0 \cdot 10^{-8}$ | 5.4 | $2.1 \cdot 10^{-9}$ | 6.0 | $8.1 \cdot 10^{-11}$ | 7.5 | $2.7 \cdot 10^{-12}$ | 8.4 |
| 8 | $6.6 \cdot 10^{-4}$ | 2.0 | $6.2 \cdot 10^{-6}$ | 3.0 | $1.0 \cdot 10^{-7}$ | 4.0 | $1.8 \cdot 10^{-9}$ | 5.1 | $3.0 \cdot 10^{-11}$ | 6.1 | $5.4 \cdot 10^{-13}$ | 7.2 | $9.1 \cdot 10^{-14}$ | * |

**Error in gradient** $|\boldsymbol{q}_h - \nabla u|_\infty$, $C_{11} = C_{22} = 0$, $\boldsymbol{C}_{12} = \boldsymbol{n}S_i^{\pm}/2$

| | $p=1$ | | $p=2$ | | $p=3$ | | $p=4$ | | $p=5$ | | $p=6$ | | $p=7$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate |
| 1 | $4.6 \cdot 10^{-2}$ | | $3.7 \cdot 10^{-3}$ | | $4.1 \cdot 10^{-4}$ | | $1.0 \cdot 10^{-4}$ | | $8.9 \cdot 10^{-6}$ | | $2.1 \cdot 10^{-6}$ | | $3.7 \cdot 10^{-7}$ | |
| 2 | $1.1 \cdot 10^{-2}$ | 1.1 | $4.3 \cdot 10^{-4}$ | 2.2 | $2.8 \cdot 10^{-5}$ | 3.2 | $2.5 \cdot 10^{-6}$ | 4.4 | $1.3 \cdot 10^{-7}$ | 5.0 | $1.5 \cdot 10^{-8}$ | 6.4 | $9.4 \cdot 10^{-10}$ | 7.3 |
| 4 | $2.7 \cdot 10^{-3}$ | 1.0 | $5.1 \cdot 10^{-5}$ | 2.1 | $1.7 \cdot 10^{-6}$ | 3.0 | $6.0 \cdot 10^{-8}$ | 4.2 | $2.1 \cdot 10^{-9}$ | 5.0 | $8.1 \cdot 10^{-11}$ | 6.4 | $2.7 \cdot 10^{-12}$ | 6.9 |
| 8 | $6.6 \cdot 10^{-4}$ | 1.0 | $6.2 \cdot 10^{-6}$ | 2.0 | $1.0 \cdot 10^{-7}$ | 2.9 | $1.8 \cdot 10^{-9}$ | 4.0 | $3.0 \cdot 10^{-11}$ | 5.0 | $5.4 \cdot 10^{-13}$ | * | $9.1 \cdot 10^{-14}$ | * |

**Error in solution** $|u_h - u|_\infty$, $C_{11} = C_{22} = 1/20$, $\boldsymbol{C}_{12} = \boldsymbol{n}S_i^{\pm}/2$

| | $p=1$ | | $p=2$ | | $p=3$ | | $p=4$ | | $p=5$ | | $p=6$ | | $p=7$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate |
| 1 | $4.5 \cdot 10^{-2}$ | | $3.9 \cdot 10^{-3}$ | | $4.9 \cdot 10^{-4}$ | | $1.2 \cdot 10^{-4}$ | | $9.3 \cdot 10^{-6}$ | | $2.5 \cdot 10^{-6}$ | | $3.8 \cdot 10^{-7}$ | |
| 2 | $1.0 \cdot 10^{-2}$ | 2.1 | $4.9 \cdot 10^{-4}$ | 3.0 | $3.7 \cdot 10^{-5}$ | 3.7 | $3.2 \cdot 10^{-6}$ | 5.2 | $1.8 \cdot 10^{-7}$ | 5.7 | $2.2 \cdot 10^{-8}$ | 6.9 | $1.1 \cdot 10^{-9}$ | 8.4 |
| 4 | $2.4 \cdot 10^{-3}$ | 2.1 | $5.8 \cdot 10^{-5}$ | 3.1 | $2.3 \cdot 10^{-6}$ | 4.0 | $7.7 \cdot 10^{-8}$ | 5.4 | $3.4 \cdot 10^{-9}$ | 5.7 | $1.2 \cdot 10^{-10}$ | 7.4 | $4.5 \cdot 10^{-12}$ | 8.0 |
| 8 | $5.9 \cdot 10^{-4}$ | 2.0 | $7.2 \cdot 10^{-6}$ | 3.0 | $1.3 \cdot 10^{-7}$ | 4.1 | $2.2 \cdot 10^{-9}$ | 5.1 | $4.5 \cdot 10^{-11}$ | 6.2 | $8.2 \cdot 10^{-13}$ | 7.2 | $1.4 \cdot 10^{-13}$ | * |

**Error in gradient** $|\boldsymbol{q}_h - \nabla u|_\infty$, $C_{11} = C_{22} = 1/20$, $\boldsymbol{C}_{12} = \boldsymbol{n}S_i^{\pm}/2$

| | $p=1$ | | $p=2$ | | $p=3$ | | $p=4$ | | $p=5$ | | $p=6$ | | $p=7$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate | Error | Rate |
| 1 | $4.5 \cdot 10^{-2}$ | | $3.9 \cdot 10^{-3}$ | | $4.9 \cdot 10^{-4}$ | | $1.2 \cdot 10^{-4}$ | | $9.3 \cdot 10^{-6}$ | | $2.5 \cdot 10^{-6}$ | | $3.8 \cdot 10^{-7}$ | |
| 2 | $1.0 \cdot 10^{-2}$ | 1.8 | $4.9 \cdot 10^{-4}$ | 2.5 | $3.7 \cdot 10^{-5}$ | 3.7 | $3.2 \cdot 10^{-6}$ | 4.6 | $1.8 \cdot 10^{-7}$ | 5.6 | $2.2 \cdot 10^{-8}$ | 6.6 | $1.1 \cdot 10^{-9}$ | 7.7 |
| 4 | $2.4 \cdot 10^{-3}$ | 1.7 | $5.8 \cdot 10^{-5}$ | 2.5 | $2.3 \cdot 10^{-6}$ | 3.7 | $7.7 \cdot 10^{-8}$ | 4.6 | $3.4 \cdot 10^{-9}$ | 5.6 | $1.2 \cdot 10^{-10}$ | 6.6 | $4.5 \cdot 10^{-12}$ | 7.6 |
| 8 | $5.9 \cdot 10^{-4}$ | 1.8 | $7.2 \cdot 10^{-6}$ | 2.6 | $1.3 \cdot 10^{-7}$ | 3.7 | $2.2 \cdot 10^{-9}$ | 4.5 | $4.5 \cdot 10^{-11}$ | 5.7 | $8.2 \cdot 10^{-13}$ | * | $1.4 \cdot 10^{-13}$ | * |

Table 2: Convergence of $u$ and $\boldsymbol{q}$ for the Poisson problem, with $C_{12} = \boldsymbol{n}S_i^{\pm}/2$ and $C_{11} = C_{22} = 0$ (top two tables), $C_{11} = C_{22} = 1/20$ (bottom two tables). For the first case we observe approximate rates of $p+1$ for $u$ and $p$ for $\boldsymbol{q}$, while the nonzero $C_{11}, C_{22}$ case appears to give a significantly higher rate for $\boldsymbol{q}$. A star symbol (*) indicates that the error is dominated by floating point rounding errors rather than the truncation error of the scheme.

### *5.2. Euler vortex*

Next we consider the compressible Euler and Navier-Stokes equations, which we write in the form:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0, \tag{44}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_i}(\rho u_i u_j + p) = +\frac{\partial \tau_{ij}}{\partial x_j} \quad \text{for } i = 1, 2, 3, \tag{45}$$

$$\frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_i}\left(u_j(\rho E + p)\right) = -\frac{\partial q_j}{\partial x_j} + \frac{\partial}{\partial x_j}(u_j \tau_{ij}), \tag{46}$$

where $\rho$ is the fluid density, $u_1, u_2, u_3$ are the velocity components, and $E$ is the total energy. The viscous stress tensor and heat flux are given by

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_j}\delta_{ij}\right) \qquad \text{and} \qquad q_j = -\frac{\mu}{\mathrm{Pr}}\frac{\partial}{\partial x_j}\left(E + \frac{p}{\rho} - \frac{1}{2}u_k u_k\right). \tag{47}$$

Here, $\mu$ is the viscosity coefficient and $\mathrm{Pr} = 0.72$ is the Prandtl number which we assume to be constant. For an ideal gas, the pressure $p$ has the form

$$p = (\gamma - 1)\rho\left(E - \frac{1}{2}u_k u_k\right), \tag{48}$$

where $\gamma$ is the adiabatic gas constant.

Our first model problem is the inviscid flow of a compressible vortex in a rectangular domain [32]. The vortex is initially centered at $(x_0, y_0)$ and is moving with the free-stream at an angle $\theta$ with respect to the $x$-axis. The analytic solution at $(x, y, t)$ is given by

$$u = u_\infty \left( \cos\theta - \frac{\epsilon((y-y_0) - \bar{v}t)}{2\pi r_c} \exp(f/2) \right), \qquad \rho = \rho_\infty \left( 1 - \frac{\epsilon^2(\gamma-1)M_\infty^2}{8\pi^2} \exp(f) \right)^{\frac{1}{\gamma-1}}, \tag{49}$$

$$v = u_\infty \left( \sin\theta + \frac{\epsilon((x-x_0) - \bar{u}t)}{2\pi r_c} \exp(f/2) \right), \qquad p = p_\infty \left( 1 - \frac{\epsilon^2(\gamma-1)M_\infty^2}{8\pi^2} \exp(f) \right)^{\frac{\gamma}{\gamma-1}}, \tag{50}$$

where $f(x, y, t) = (1 - ((x-x_0) - \bar{u}t)^2 - ((y-y_0) - \bar{v}t)^2)/r_c^2$, $M_\infty$ is the Mach number, $\gamma = c_p/c_v = 1.4$, and $u_\infty$, $p_\infty$, $\rho_\infty$ are free-stream velocity, pressure, and density. The Cartesian components of the free-stream velocity are $\bar{u} = u_\infty \cos\theta$ and $\bar{v} = u_\infty \sin\theta$. The parameter $\epsilon$ measures the strength of the vortex and $r_c$ is its size.

We use a domain of size 20-by-15, with the vortex initially centered at $(x_0, y_0) = (5, 5)$ with respect to the lower-left corner. The Mach number is $M_\infty = 0.5$, the angle $\theta = \arctan 1/2$, and the vortex has the parameters $\epsilon = 0.3$ and $r_c = 1.5$. We use characteristic boundary conditions and integrate until time $t_0 = \sqrt{10^2 + 5^2}/10$, when the vortex has moved a relative distance of $(1, 1/2)$.

We write the Euler equations as a first-order system of conservation laws (1), in the conserved variables $(\rho, \rho u, \rho v, \rho E)$. The scheme (10) is implemented in a straight-forward way, and we use Roe's method for the numerical fluxes (5) [33]. The time-integration is done explicitly with the form (36) using the RK4 solver and a timestep $\Delta t$ small enough so that all truncation errors are dominated by the spatial discretization. We start from a coarse unstructured quadrilateral mesh (figure 5, top left), which we refine uniformly a number of times, and we use polynomial degrees $p$ ranging between 1 and 8. The top right plot also shows the density field for a sample solution.

In the bottom plot of figure 5, we graph the maximum errors (discretely at the solution nodes) for all simulation cases, both for the Line-DG method and the standard nodal DG method. The results clearly show the optimal order of convergence $\mathcal{O}(h^{p+1})$ for element size $h$ for both methods, and that the Line-DG errors are in all cases very close to those of the nodal DG method.

### *5.3. Inviscid flow over a cylinder*

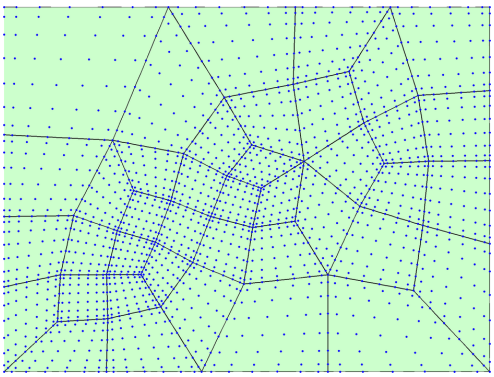
Next we study a problem with a steady-state solution and curved boundaries, and solve the Euler equations for the inviscid flow over a half-cylinder with radius 1 at a Mach number of 0.3. Structured quadrilateral meshes are used, with strong element size grading to better resolve the region close to the cylinder (see figure 6, top left). The outer domain boundary is a half-cylinder with radius 10, where characteristic boundary conditions are imposed. Standard slip wall/symmetry conditions are used at the cylinder and at the symmetry plane.

The steady-state solutions are found using a fully consistent Newton method, applied directly to the equations (10), with the linear systems solved using a direct sparse solver [25]. Starting the iterations from an approximate analytical solution, derived from a potential flow approximation, the solver converges to machine precision in 4 to 6 iterations. The solution is shown in the bottom left of figure 6, and the figures to the right show portions of the Jacobian matrices for both the Line-DG and the nodal DG method. This illustrates again the reduced sparsity of the Line-DG scheme, with about a factor of 4 fewer entries than nodal DG already in two space dimensions.

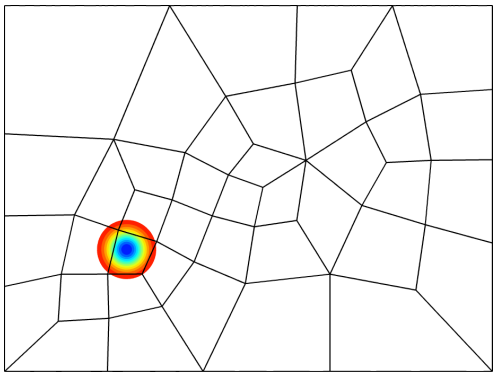
To evaluate the accuracy and convergence of the scheme, in figure 7 we plot the errors in the lift coefficient $C_L$ (left) and the maximum errors in the entropy (right). These plots again confirm the convergence of the schemes as well as the minor differences in error between the Line-DG and the nodal DG schemes.

### *5.4. Laminar flow around airfoil*

An example of a steady-state viscous computation is shown in figure 8. The compressible Navier-Stokes equations are solved at Mach 0.2 and Reynolds number 5000, for a flow around an SD7003 airfoil. The



Coarsest mesh, with degree $p = 7$



Solution (density)

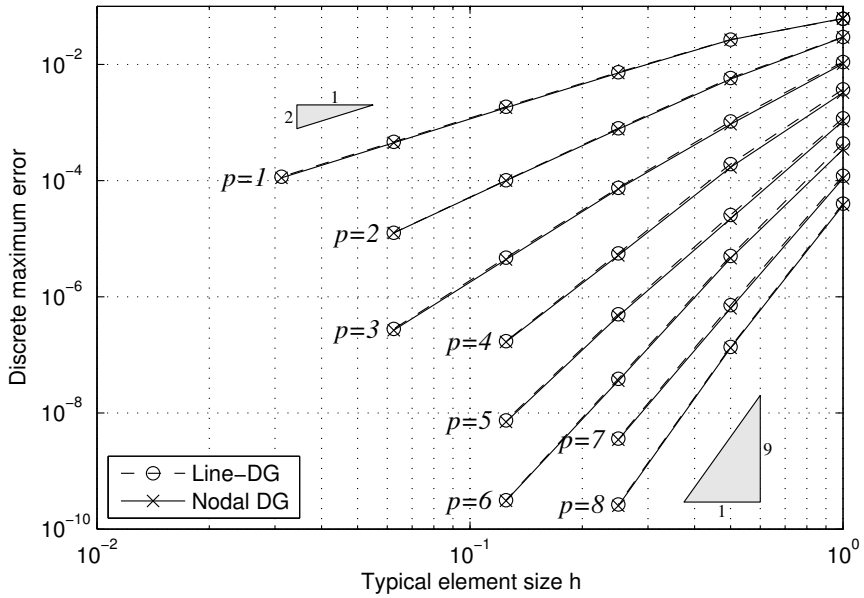



Figure 5: Convergence test for an Euler vortex test problem using the Line-DG method and the nodal DG method. The results show optimal order of convergence $\mathcal{O}(h^{p+1})$ with very small differences between the two methods.

quadrilateral mesh is fully unstructured except for a structured graded boundary layer region, with a total of 461 elements for the coarse mesh, and 1844 and 7376 elements for the once and twice refined meshes, respectively. With approximating polynomials of degree $p = 6$, this gives a total number of high-order nodes of 22,589 for the coarse mesh and 90,356 for the first refinement.

We find the steady-state solution with 10 digits of accuracy in the residual using a consistent Newton's method, with pseudo-timestepping for regularization. A solution is shown in figure 8 (top right), for the coarse mesh with $p = 7$. In the bottom plots, we show the convergence of the drag and the lift coefficients, for a range of polynomial degrees $p$. While it is hard to asses the exact order of convergence from these numbers, it is clear that our scheme provides a high order of convergence even for these difficult derivative-based quantities.

### *5.5. Transient flow around airfoil at Re = 20,000*

In our last example, we demonstrate time-accurate implicit solution of transient flow around an SD7003 airfoil at Re = 20,000. The mesh is highly resolved in the boundary layer, however, for this Reynolds

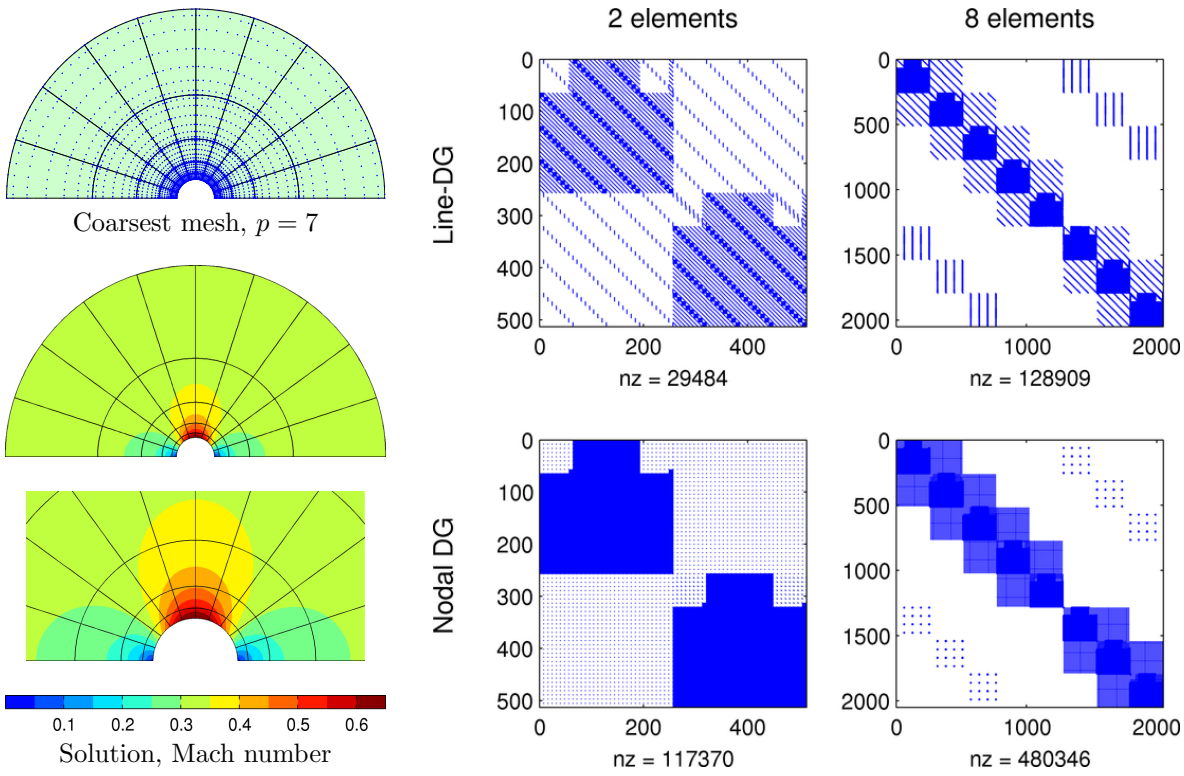



Figure 6: Inviscid flow over a cylinder. The plots show the coarsest grid used in the convergence study and the nodes for polynomial degree $p = 7$ (top left), the corresponding solution as Mach number color plot (bottom left, with zoom-in), and a sparsity plot of the Jacobian matrices for both Line-DG and nodal DG (right).

number it is coarser than the flow features in much of the domain and the computations should therefore be considered an under-resolved ILES-type model [34].

The Mach number is 0.1 and the angle of attack is 30 degrees to force flow separation at the leading edge. We use the three-stage DIRK scheme (37), (38), solved with Newton's method as described in sections 4.2 and 4.3. At each Newton step we perform 5 GMRES iterations, and if the number of Newton iterations exceeds 15 we recompute the Jacobian matrices. Our computational mesh has 1122 quadrilateral elements with polynomial degrees $p = 7$. The mesh and a solution at the normalized time of $t = 1.76$ are shown in the top plots of figure 9.

We use the timestep $\Delta t = 2 \cdot 10^{-4}$, which is about 250 times larger than the largest stable explicit RK4 timestep, yet small enough to accurately capture most of the complex flow features. It is difficult to estimate the accuracy in the simulation, due to the under-resolved nature of LES and the high sensitivity of transitional flows. However, we have run the same problem using a nodal DG code which has been tested against other simulations as well as experiments [34]. The bottom left plot of figure 9 shows that the lift and drag forces on the airfoil agree well between the two schemes, until small perturbations have grown enough to cause large differences between the flows.

We also plot the performance of the Newton solver, in the bottom right of figure 9. It shows how the number of Newton iterations per solve remains fairly constant, and about once every 100th timestep it reaches 16 which forces a recomputation of the Jacobian matrix. The average number of iterations per Newton solve for the entire simulation is 14. Because of the sparsity of the matrices and the splitting (41), these iterations are relatively inexpensive compared to residual evaluation. Our implementation is not optimized for performance, but the relative times for the four operations (1) GMRES iteration, (2) residual evaluation, (3) Jacobian evaluation, and (4) preconditioner factorization are roughly 1:4:40:16. Therefore,

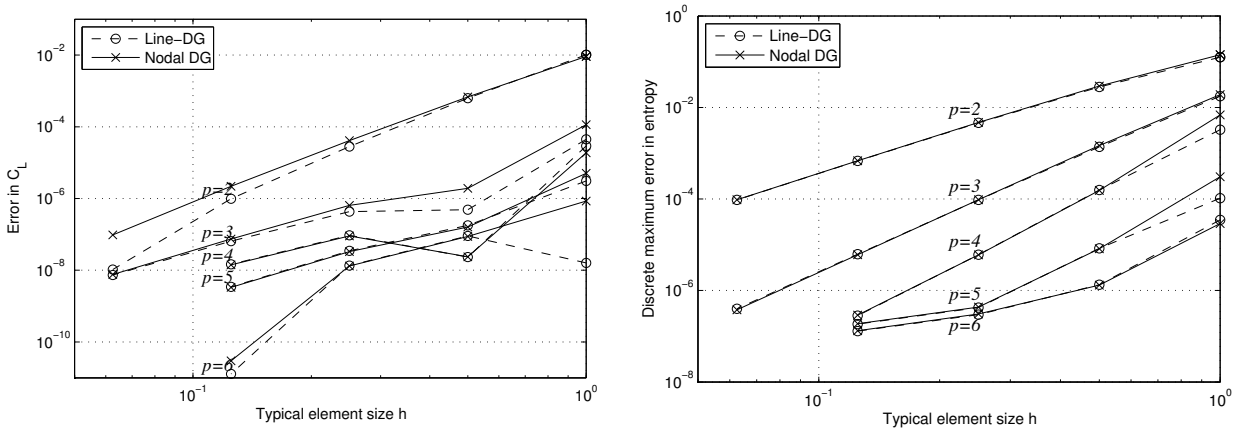



Figure 7: The convergence of the lift coefficient $C_L$ (left) and the entropy difference (right) for the inviscid flow over cylinder problem. The plots show a series of results for varying polynomial degrees and number of refinements, for the two methods Line-DG and nodal DG.

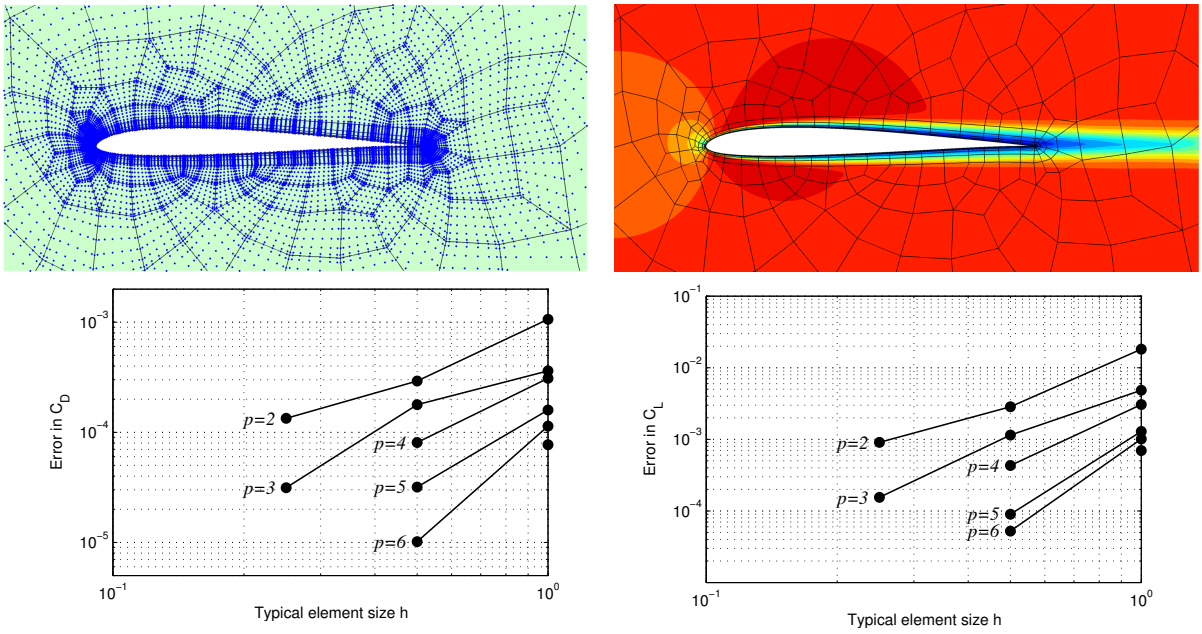



Figure 8: Stationary flow around an SD7003 airfoil (top left: mesh, top right: Mach number), computed with the Line-DG method with $p = 7$, at free-stream Mach 0.2, zero angle of attack, and Reynolds number 5,000. The bottom plots show the convergence of $C_D$ and $C_L$, for a range of polynomial degrees and with 0, 1, or 2 uniform mesh refinements.

since we only perform 5 GMRES iterations per solve, we spend about the same time in residual evaluation as in the linear solver. In this sense, the solver is similar to an explicit scheme in that it spends a large portion of its computational time in residual evaluations, which gives benefits e.g. in the parallelization on new parallel multicore computer architectures with limited memory bandwidth. The time for re-assembly and factorization of the preconditioner is negligible, since they are only performed once in about every 100th timestep, which corresponds to 1400 residual evaluations or 7000 GMRES iterations. We also point out that without preconditioning about 20 times more GMRES iterations are required for the same tolerance, showing that even our simple preconditioner makes a drastic difference on the convergence.

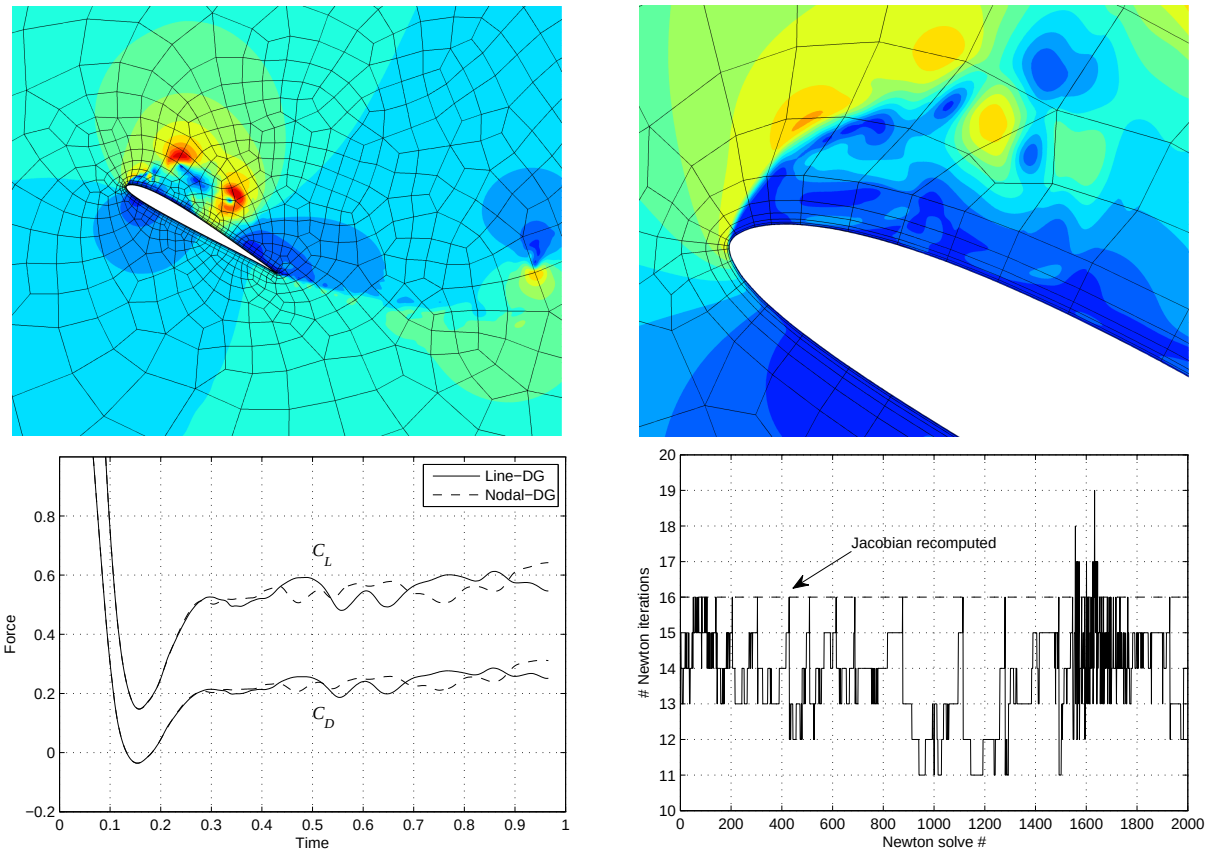



Figure 9: Implicit transient simulation of flow around an SD7003 airfoil, at 30 degrees angle of attack, Reynolds number 20,000, and Mach number 0.1. Line-DG with polynomial degrees of $p = 7$ is used for the spatial discretization, and a three stage, third order accurate DIRK scheme is used for time integration. The nonlinear systems are solved using a Newton-Krylov solver with re-used Jacobian matrices. The CFL number compared to the explicit RK4 method is about 250. The top figures show the computational mesh and color contours of the Mach number, from 0 to 0.3. The bottom left plot compares the lift and drag coefficient with a nodal DG scheme, indicating good agreement until the small differences between the schemes have grown enough to make the solutions hard to compare. The bottom right plot shows the performance of the nonlinear Newton solver, and in particular how it only recomputes the Jacobian matrices once in about every 100th solve.

## 6. Conclusions

We have presented a new line-based DG method for first and second-order systems of equations. The scheme has a simple structure, with only one-dimensional integrals and standard Riemann solvers applied point-wise. Compared to the standard nodal DG method, this gives a simpler assembly process and a fundamentally different sparsity structure, which we used to develop efficient matrix-based implicit solvers. Compared to collocation based methods such as the DG spectral element method, it uses fully consistent integration along each coordinate-direction, and it slightly reduces the connectivities to neighboring elements by the choice of solution nodes. We showed that the accuracy of the discretizations are very similar to the standard DG method, and we demonstrated a stiff LES-type flow simulation with high-order DIRK time-integration with Newton-Krylov solvers and re-used Jacobians.

A number of further developments are needed to make the scheme competitive for real-world problems. We have not addressed the issue of nonlinear stability for under-resolved features, including shock capturing, where approaches such as artificial viscosity and limiting could be adopted. For the solvers, our simple block-Jacobi preconditioner can be much improved upon, using e.g. multigrid and ILU techniques. Finally, for large problems the implementation needs to be parallelized, in particular for the new generation of multicore and GPU chips where memory bandwidth is limited. Here the high sparsity of the Line-DG scheme might

have additional benefits over the standard nodal DG scheme.

## 7. Acknowledgments

We would like to acknowledge all the valuable discussions about this work with Jaime Peraire, Luming Wang, and Bradley Froehle, the suggestions from the reviewers, as well as the generous support from the AFOSR Computational Mathematics program under grant FA9550-10-1-0229, the Alfred P. Sloan foundation, and the Director, Office of Science, Computational and Technology Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## References

[1] S. K. Lele, Compact finite difference schemes with spectral-like resolution, J. Comput. Phys. 103 (1992) 16–42.
[2] M. R. Visbal, D. V. Gaitonde, On the use of higher-order finite-difference schemes on curvilinear and deforming meshes, J. Comput. Phys. 181 (2002) 155–185.
[3] J. Nordström, J. Gong, E. van der Weide, M. Svärd, A stable and conservative high order multi-block method for the compressible Navier-Stokes equations, J. Comput. Phys. 228 (2009) 9020–9035.
[4] T. J. Barth, Recent developments in high order k-exact reconstruction on unstructured meshes, AIAA 31st Aerospace Sciences Meeting (1993).
[5] A. Nejat, C. Ollivier-Gooch, A high-order accurate unstructured finite volume Newton-Krylov algorithm for inviscid compressible flows, J. Comput. Phys. 227 (2008) 2582–2609.
[6] T. J. R. Hughes, G. Scovazzi, T. E. Tezduyar, Stabilized methods for compressible flows, J. Sci. Comput. 43 (2010) 343–368.
[7] W. H. Reed, T. R. Hill, Triangular mesh methods for the neutron transport equation, Technical Report Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.
[8] B. Cockburn, C.-W. Shu, Runge-Kutta discontinuous Galerkin methods for convection-dominated problems, J. Sci. Comput. 16 (2001) 173–261.
[9] J. S. Hesthaven, T. Warburton, Nodal discontinuous Galerkin methods, volume 54 of *Texts in Applied Mathematics*, Springer, New York, 2008. Algorithms, analysis, and applications.
[10] D. A. Kopriva, J. H. Kolias, A conservative staggered-grid Chebyshev multidomain method for compressible flows, J. Comput. Phys. 125 (1996) 244–261.
[11] Z. J. Wang, Spectral (finite) volume method for conservation laws on unstructured grids. Basic formulation, J. Comput. Phys. 178 (2002) 210–251.
[12] Y. Liu, M. Vinokur, Z. J. Wang, Spectral difference method for unstructured grids. I. Basic formulation, J. Comput. Phys. 216 (2006) 780–801.
[13] H. Huynh, A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods, in: 18th AIAA Computational Fluid Dynamics Conference, Miami, Florida. AIAA-2007-4079.
[14] P. E. Vincent, P. Castonguay, A. Jameson, A new class of high-order energy stable flux reconstruction schemes, J. Sci. Comput. 47 (2011) 50–72.
[15] B. Cockburn, B. Dong, J. Guzmán, A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems, Math. Comp. 77 (2008) 1887–1916.
[16] J. Peraire, N. C. Nguyen, B. Cockburn, An implicit high-order hybridizable discontinuous Galerkin method for linear convection-diffusion equations, J. Comput. Phys. 228 (2009) 3232–3254.
[17] D. A. Kopriva, G. Gassner, On the quadrature and weak form choices in collocation type discontinuous Galerkin spectral element methods, J. Sci. Comput. 44 (2010) 136–155.
[18] A. M. Fernando, F. Q. Hu, DGM-FD: a finite difference scheme based on the discontinuous Galerkin method applied to wave propagation, J. Comput. Phys. 230 (2011) 4871–4898.
[19] T. Haga, H. Gao, Z. Wang, A high-order unifying discontinuous formulation for 3D mixed grids, in: 48th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, Florida. AIAA-2010-540.
[20] B. Cockburn, C.-W. Shu, The local discontinuous Galerkin method for time-dependent convection-diffusion systems, SIAM J. Numer. Anal. 35 (1998) 2440–2463 (electronic).
[21] P.-O. Persson, J. Peraire, Curved mesh generation and mesh refinement using Lagrangian solid mechanics, in: 47th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, Florida. AIAA-2009-949.
[22] D. N. Arnold, F. Brezzi, B. Cockburn, L. D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, SIAM J. Numer. Anal. 39 (2001/02) 1749–1779 (electronic).
[23] J. Peraire, P.-O. Persson, The compact discontinuous Galerkin (CDG) method for elliptic problems, SIAM J. Sci. Comput. 30 (2008) 1806–1824.
[24] B. Cockburn, B. Dong, An analysis of the minimal dissipation local discontinuous Galerkin method for convection-diffusion problems, J. Sci. Comput. 32 (2007) 233–262.
[25] T. A. Davis, Direct methods for sparse linear systems, volume 2 of *Fundamentals of Algorithms*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.
[26] R. Alexander, Diagonally implicit Runge-Kutta methods for stiff o.d.e.'s, SIAM J. Numer. Anal. 14 (1977) 1006–1021.

[27] P.-O. Persson, J. Peraire, Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations, SIAM J. Sci. Comput. 30 (2008) 2709–2733.
[28] S. A. Orszag, Spectral methods for problems in complex geometries, J. Comput. Phys. 37 (1980) 70–92.
[29] C. Canuto, P. Pietra, Boundary and interface conditions within a finite element preconditioner for spectral methods, J. Comput. Phys. 91 (1990) 310–343.
[30] C. Rumsey, M. Sanetrik, R. Biedron, N. Melson, E. Parlette, Efficiency and accuracy of time-accurate turbulent Navier-Stokes computations, in: 13th AIAA Applied Aerodynamics Conference, San Diego, California. AIAA-95-1835.
[31] Y. Saad, M. H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.
[32] G. Erlebacher, M. Y. Hussaini, C.-W. Shu, Interaction of a shock with a longitudinal vortex, J. Fluid Mech. 337 (1997) 129–153.
[33] P. L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys. 43 (1981) 357–372.
[34] A. Uranga, P.-O. Persson, M. Drela, J. Peraire, Implicit large eddy simulation of transition to turbulence at low reynolds numbers using a discontinuous galerkin method, Int. J. Num. Meth. Eng. 87 (2011) 232–261.