SIAM J. SCI. COMPUT.
Vol. 30, No. 6, pp. 2709–2733

# NEWTON-GMRES PRECONDITIONING FOR DISCONTINUOUS GALERKIN DISCRETIZATIONS OF THE NAVIER–STOKES EQUATIONS*

P.-O. PERSSON† AND J. PERAIRE‡

**Abstract.** We study preconditioners for the iterative solution of the linear systems arising in the implicit time integration of the compressible Navier–Stokes equations. The spatial discretization is carried out using a discontinuous Galerkin method with fourth order polynomial interpolations on triangular elements. The time integration is based on backward difference formulas resulting in a nonlinear system of equations which is solved at each timestep. This is accomplished using Newton's method. The resulting linear systems are solved using a preconditioned GMRES iterative algorithm. We consider several existing preconditioners such as block Jacobi and Gauss–Seidel combined with multilevel schemes which have been developed and tested for specific applications. While our results are consistent with the claims reported, we find that these preconditioners lack robustness when used in more challenging situations involving low Mach numbers, stretched grids, or high Reynolds number turbulent flows. We propose a preconditioner based on a coarse scale correction with postsmoothing based on a block incomplete LU factorization with zero fill-in (ILU0) of the Jacobian matrix. The performance of the ILU0 smoother is found to depend critically on the element numbering. We propose a numbering strategy based on minimizing the discarded fill-in in a greedy fashion. The coarse scale correction scheme is found to be important for diffusion dominated problems, whereas the ILU0 preconditioner with the proposed ordering is effective at handling the convection dominated case. While little can be said in the way of theoretical results, the proposed preconditioner is shown to perform remarkably well for a broad range of representative test problems. These include compressible flows ranging from very low Reynolds numbers to fully turbulent flows using the Reynolds averaged Navier–Stokes equations discretized on highly stretched grids. For low Mach number flows, the proposed preconditioner is more than one order of magnitude more efficient than the other preconditioners considered.

**Key words.** discontinuous Galerkin, implicit solvers, iterative solvers, GMRES, ILU factorization, multigrid

**AMS subject classifications.** 65M60, 65N22, 65F10

**DOI.** 10.1137/070692108

## 1. Introduction.

Discontinuous Galerkin (DG) methods using high order approximations have become an attractive alternative for the solution of systems of conservation laws [22, 9, 32]. The ability to obtain very accurate spatial discretizations on arbitrary unstructured meshes makes them particularly suited for wave propagation problems in which low dispersion errors are a requirement. One of the attractive features of DG methods is the very natural treatment of the convective operators through the stabilizing interelement jump terms. This avoids the need for cumbersome interior stabilization terms. On the other hand, these advantages come at the expense of an increased number of degrees of freedom. For explicit time integration schemes, the penalty associated with the additional degrees of freedom is often not significant and is offset by the added benefits of DG methods. Unfortunately, the number of realistic problems which are amenable to explicit solution is very small.

*Received by the editors May 16, 2007; accepted for publication (in revised form) October 9, 2007; published electronically October 13, 2008.
http://www.siam.org/journals/sisc/30-6/69210.html

†Department of Mathematics, MIT, 77 Massachusetts Avenue 2-363A, Cambridge, MA 02139 (persson@mit.edu).

‡Department of Aeronautics and Astronautics, MIT, 77 Massachusetts Avenue 37-451, Cambridge, MA 02139 (peraire@mit.edu).

In common situations, the large variations in element size required to resolve the multiple spatial scales occurring in high Reynolds number flows render the use of explicit time integration techniques impractical. Also, systems of equations involving multiple timescales, such as those resulting from low Mach number flows, necessitate the use of implicit integration methods.

For general nonlinear systems of equations, the implicit time discretization formulas lead to nonlinear algebraic systems of equations which need to be solved at each timestep. Clearly, the ability to solve these large systems is a critical step in rendering the use of DG methods feasible. Furthermore, for large problems, particularly in three dimensions, it is too expensive to use direct solution techniques, and therefore, iterative methods must be employed. The development of iterative methods for convective-diffusive systems has been a topic of considerable interest in the past. More recently, some solution strategies have been proposed which are specific to DG methods. It turns out that while some concepts are applicable independently of the discretization method used, DG discretizations possess a certain block structure which sets them apart from other alternative discretization methods.

In [17], the use of a block Gauss–Seidel (GS) smoother in combination with a multigrid method is presented and shown to possess optimal behavior for linear advective diffuse systems resulting from DG discretizations. The performance of the GS smoother is found to depend critically on the element ordering. For simple problems, obtaining good orderings using heuristic arguments is feasible, but the situation is more complex for systems of equations such the Navier–Stokes equations. In [31] linear and nonlinear multigrid methods for DG algorithms are presented using block Jacobi and block GS smoothers. Good performance is demonstrated for the Euler equations on moderate subsonic Mach numbers and fairly isotropic meshes. In [14] a multigrid approach for DG discretizations of the Euler and laminar Navier–Stokes equations is described. They show that by using a smoother based on the solution of block tridiagonal systems significant performance improvements can be obtained relative to the simple block Jacobi smoother. Similar ideas were also used earlier in the context of finite volume solvers in [29]. In general, these methods show promise and put forward some of the ingredients which appear to be required for a scalable robust DG solution method. In particular, the use of blocks rather than individual components in the construction of the smoothers seems to be a requirement to eliminate high $p$ dependency [17] and is a natural choice in the DG context. For the pure convective problem, there is little to be gained by using multilevel methods, and in this limit, the use of GS or line relaxation methods with suitable orderings is optimal. On the other hand, multilevel strategies [6, 5] are necessary to attain efficiency for elliptic-type phenomena.

In this paper, we assess various preconditioning techniques from the point of view of robustness and performance consistency over a broad range of realistic problems. While our findings are largely consistent with the results published, we find that the methods described above lack robustness and are not generally applicable to many realistic engineering problems.

We consider implicit time discretization procedures, based on a backward difference approximation of the time derivative. The space discretization of the governing equations is carried out using the DG method on triangular meshes. Here, the viscous terms are discretized using the compact DG (CDG) method [34], but we expect that the algorithms discussed would be equally applicable to other types of viscous discretization schemes. The nonlinear system of equations generated at each timestep is linearized exactly and solved using Newton's method. We are interested in high order spatial approximations, and the approach described here is general. However, for simplicity we present all our results using a fourth order approximation of the solution unknowns. From our own experience, this appears to be a good compromise between cost and accuracy for the class of problems we consider.

The linear systems generated in the Newton process are solved by iteration. Here, we consider a GMRES method and think of the multilevel solver as a preconditioner. This is a rather common strategy for the solution of complex problems and is credited with making the overall method more robust at a small additional expense [29, 17, 26, 25]. We consider several preconditioning options: block Jacobi, block GS, and block incomplete LU factorizations with zero fill-in (ILU0). For the GS and ILU0 methods, we propose inexpensive ordering algorithms which are based on minimizing the magnitude of the discarded fill-in in a greedy fashion. While incomplete LU factorizations are commonly used as preconditioners for the GMRES algorithm [38, 25] or smoothers for multigrid schemes [23, 41, 42], the use of block ILU0 as a smoother for DG discretizations was only introduced in [35]. If the triangular meshes considered are such that the neighbors of an element do not neighbor each other, the $U$ factor in the ILU0 factorization is identical to the upper triangular part of the unfactored Jacobian matrix. This property extends to three dimensions and justifies regarding the GS method as a particular ILU0 factorization in which the lower factor $L$ is taken to be zero. In the diffusive limit, we find it necessary to introduce a coarse grid correction. In this case, the block Jacobi, block GS, or block ILU0 are used as postsmoothers after the coarse grid correction is calculated. We restrict our attention to single coarse grid corrections obtained by using lower order approximations on each element. In particular, we consider $p = 0$ (piecewise constant) or $p = 1$ (piecewise linear) approximations. We have found little theoretical and empirical justification for employing $p$-multigrid strategies [37, 14] in this context given that, by retaining the block structure, the local elemental problems on the fine mesh are solved exactly. For the problems considered here, the coarse grid correction is solved exactly since it results in small problems, but for larger problems a scalable multilevel solution of the coarse grid problem should be considered.

The combined ILU0 preconditioner with either $p = 0$ or $p = 1$ coarse grid correction appears to outperform all the other options considered. It shows a remarkable consistency and robustness over a range of test cases including flows ranging from very low Reynolds numbers to fully turbulent flows using the Reynolds averaged Navier–Stokes equations discretized on highly stretched grids. We acknowledge that in the general case there is very little that can be proven rigorously. Nevertheless, the approaches we propose seem to perform reliably over a very broad range of problems and we believe they have a fairly general applicability. A distinctive feature of the proposed preconditioner relative to the other alternatives is its behavior for low Mach number flows. For ILU0 with $p = 1$ coarse grid correction we experience convergence behavior which is essentially independent of the Mach number. In this scenario, the proposed preconditioner is more than one order of magnitude more efficient than the other approaches considered. Finally, for time accurate solutions, we also consider the effect of the preconditioners as a function of the time step employed in the time integration.

In this paper, we do not consider parallel implementation issues even though it is clear that practical uses of DG methods will require parallel computing. We recognize the fact that parallelizing LU factorizations is not trivial, but we expect that our ordering algorithms can guide the selection of efficient partitions. One approach for parallelization of ILU is proposed in [20], and related work on domain decomposition and approximate factorizations includes [13, 18, 39]. Iterative solvers of ILU and multigrid type are also used in many parallel solver libraries, such as PETSc [1] and AztecOO [21].

## 2. Problem formulation.

**2.1. Equations and discretization.** We consider a time-dependent system of conservation laws of the form

$$\frac{\partial u}{\partial t} + \nabla \cdot F_i(u) - \nabla \cdot F_v(u, \nabla u) = S(u, \nabla u) \tag{2.1}$$

in a domain $\Omega$, with conserved state variables $u$, inviscid flux function $F_i$, viscous flux function $F_v$, and source term $S$. We allow for the inviscid and viscous fluxes as well as the source term to be nonlinear functions of their arguments. In order to develop a DG discretization of this problem, we eliminate the second order spatial derivatives of $u$ by introducing additional variables $q = \nabla u$. Thus, the original system of equations (2.1) is now rewritten as

$$\frac{\partial u}{\partial t} + \nabla \cdot F_i(u) - \nabla \cdot F_v(u, q) = S(u, q), \tag{2.2}$$

$$q - \nabla u = 0. \tag{2.3}$$

Next, we consider a triangulation $\mathcal{T}_h$ of the spatial domain $\Omega$ and introduce the finite element spaces $V_h$ and $\Sigma_h$ as

$$V_h = \{v \in [L^2(\Omega)]^m \mid v|_K \in [\mathcal{P}_p(K)]^m \quad \forall K \in \mathcal{T}_h\}, \tag{2.4}$$

$$\Sigma_h = \{r \in [L^2(\Omega)]^{dm} \mid r|_K \in [\mathcal{P}_p(K)]^{dm} \quad \forall K \in \mathcal{T}_h\}, \tag{2.5}$$

where $\mathcal{P}_p(K)$ is the space of polynomial functions of degree at most $p \geq 0$ on triangle $K$, $m$ is the dimension of $u$, and $d$ is the spatial dimension.

We now consider DG formulations of the following form: find $u_h \in V_h$ and $q_h \in \Sigma_h$ such that for all $K \in \mathcal{T}_h$, we have

$$\int_K q_h \cdot r \, dx = -\int_K u_h \nabla \cdot r \, dx + \int_{\partial K} \hat{u} r \cdot n \, ds \quad \forall r \in [\mathcal{P}_p(K)]^{dm}, \tag{2.6}$$

$$\begin{aligned} &\int_K \frac{\partial u_h}{\partial t} v \, dx - \int_K [F_i(u_h) - F_v(u_h, q_h)] \cdot \nabla v \, dx \\ &\quad = \int_K S(u_h, q_h) v \, dx - \int_{\partial K} [\hat{F}_i - \hat{F}_v] \cdot n v \, ds \quad \forall v \in [\mathcal{P}_p(K)]^m. \end{aligned} \tag{2.7}$$

Here, the numerical fluxes $\hat{F}_i$, $\hat{F}_v$, and $\hat{u}$ are approximations to $F_i$, $F_v$, and $u$, respectively, on the boundary $\partial K$ of the element $K$. The DG formulation is complete once these numerical fluxes are specified in terms of $q_h$, $u_h$, and the boundary conditions.

In order to ensure conservation, the normal component of both the viscous and inviscid numerical fluxes at the element boundaries is chosen to be continuous across elements. $\hat{u}$ is also continuous across elements in our implementation. In particular, the inviscid flux is determined using Roe's scheme [36], whereas the viscous flux is calculated using the CDG method [34].

We note that if the numerical flux $\hat{u}$ is chosen to be a function of $u_h$ and not $q_h$, then the additional $q_h$ variables can be eliminated after discretization at element level, thus, resulting in a system involving only the degrees of freedom corresponding to the conserved variables $u_h$. The final result is a system of coupled ordinary differential equations (ODEs) of the form

$$\boldsymbol{M}\dot{\boldsymbol{u}} = \boldsymbol{R}(\boldsymbol{u}), \tag{2.8}$$

where $\boldsymbol{u}(t)$ is a vector containing the degrees of freedom associated with $u_h$, and $\dot{\boldsymbol{u}}$ denotes the time derivative of $\boldsymbol{u}(t)$. Here, $\boldsymbol{M}$ is the mass matrix and $\boldsymbol{R}$ is the residual vector which is a nonlinear function of $\boldsymbol{u}$. We use nodal basis expansions [22] to represent $u_h$ inside each element.

Given an initial condition $\boldsymbol{u}(0) = \boldsymbol{u}_0$, the system of ODEs (2.8) needs to be further integrated in time. Explicit techniques such as the popular fourth order Runge–Kutta scheme have timestep size restrictions that are too severe for many applications of interest, and therefore, we consider implicit solution techniques.

The backward differentiation formula [40] of order $k$ (BDF-$k$) approximates the time derivative at time $t_n = n\Delta t$ in terms of $\boldsymbol{u}(t_n)$ and the solution at $k$ previous timesteps: $\dot{\boldsymbol{u}} \approx \frac{1}{\Delta t}\sum_{i=0}^{k} \alpha_i \boldsymbol{u}_{n-i}$. Here, $\boldsymbol{u}_n$ denotes an approximation to $\boldsymbol{u}(t_n)$ and $\Delta t$ is the timestep. The system of equations that needs to be solved in order to compute $\boldsymbol{u}_n$ at each time level then becomes

$$\boldsymbol{M}\sum_{i=0}^{k} \alpha_i \boldsymbol{u}_{n-i} - \Delta t \boldsymbol{R}(\boldsymbol{u}_n) = 0. \tag{2.9}$$

We use a damped Newton method to solve these equations. An initial guess $\boldsymbol{u}_n^{(0)}$ is formed by extrapolating from the $k$ previous solutions, and iterates $\boldsymbol{u}_n^{(j)}$ are evaluated by computing corrections according to the linearized equation

$$\boldsymbol{J}(\boldsymbol{u}_n^{(j)})\Delta\boldsymbol{u}_n^{(j)} = \boldsymbol{R}_{\mathrm{BDF}}(\boldsymbol{u}_n^{(j)}), \tag{2.10}$$

where the BDF nonlinear residual is given by

$$\boldsymbol{R}_{\mathrm{BDF}}(\boldsymbol{u}_n) = \boldsymbol{M}\sum_{i=0}^{k} \alpha_i \boldsymbol{u}_{n-i} - \Delta t \boldsymbol{R}(\boldsymbol{u}_n) \tag{2.11}$$

and the Jacobian $J$ is obtained by differentiation of the BDF residual. Thus, dropping the iteration superscript, we have

$$\boldsymbol{J}(\boldsymbol{u}_n) = \frac{d\boldsymbol{R}_{\mathrm{BDF}}}{d\boldsymbol{u}_n} = \alpha_0\boldsymbol{M} - \Delta t\frac{d\boldsymbol{R}}{d\boldsymbol{u}_n} \equiv \alpha_0\boldsymbol{M} - \Delta t\boldsymbol{K}. \tag{2.12}$$

The new iterates are obtained as $\boldsymbol{u}_n^{(j+1)} = \boldsymbol{u}_n^{(j)} + \beta\Delta\boldsymbol{u}_n^{(j)}$, where $\beta$ is a damping parameter which is determined by forcing the residual to decrease after each iteration [33]. These iterations are continued until the residual is sufficiently small.

In this paper, we will focus on the solution of the linear system of equations (2.10) using iterative methods and deliberately ignore other issues related to the convergence of the Newton method, such as the termination criterion for the Newton iterations. We will study several representative test problems in detail for a range of timesteps $\Delta t$ and other problem parameters. The constant $\alpha_0$ is assumed to be exactly one for simplicity (which is the case for $k = 1$ and a good approximation for higher $k$).

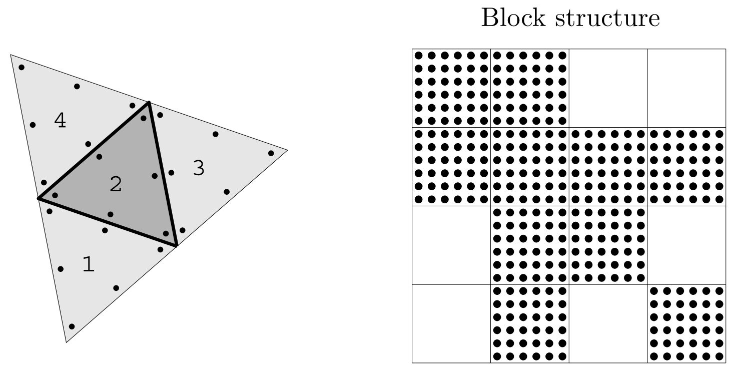


FIG. 2.1. *Sparsity structure of the matrix $\boldsymbol{A}$ for four triangles with $p = 2$, and $m = 1$.*

**2.2. Jacobian sparsity pattern and representation.** The system matrix $\boldsymbol{A} = \boldsymbol{M} - \Delta t\boldsymbol{K}$ is sparse, and its structure depends on the ordering of the unknowns within the vector $\boldsymbol{u}$. Since in the DG setting each unknown can be unambiguously associated to an element, it seems natural to order the components of $\boldsymbol{u}$ in such a way that all the unknowns associated to an element appear contiguously. Furthermore, it is computationally efficient to number all the unknowns corresponding to each of the $m$ components of $u$ consecutively within each element. This allows common operations required for the residual and Jacobian evaluation, such as interpolating the unknowns at the integration points, to be carried out while maximizing contiguous memory access. With this ordering, the matrix $\boldsymbol{A}$ has a block structure. In particular, $\boldsymbol{M}$ is block diagonal as it involves no dependencies between elements or solution components. The size of each block is given by the number of nodes within each element, and the total number of blocks is equal to the number of elements times $m$. On the other hand, the matrix $\boldsymbol{K}$ has a nontrivial block structure which is determined by the element connectivity. The size of each block is equal to the number of nodes within each element times $m$. We note that all the nonzero entries in $\boldsymbol{M}$ are also nonzero entries in $\boldsymbol{K}$, and therefore the sparsity pattern of $\boldsymbol{A}$ is determined by $\boldsymbol{K}$.

In principle, the matrix $\boldsymbol{A}$ could be represented in a general sparse matrix format, such as the compressed column format [2], but that would make it harder to take advantage of the large dense blocks. Instead, we use a dense block format. Furthermore, when using triangles in two dimensions and tetrahedra in three dimensions the number of nonzero blocks in each row, except for boundary elements, is known a priori (four in two dimensions and five in three dimensions). For illustration purposes, Figure 2.1 shows the sparsity pattern of the $\boldsymbol{A}$ matrix for a sample mesh consisting of four triangular elements and $m = 1$. We note that this pattern is common to many DG methods for elliptic operators such as the interior point method [11], the scheme of Bassi and Rebay [3], or the CDG scheme used here [34]. However, schemes such as the local DG [8] are known to be noncompact and therefore have a larger number of nonzero blocks.

To illustrate the high cost associated with storing matrices for high order DG discretizations, we consider a typical compressible Navier–Stokes problem in three dimensions with a one-equation turbulence model. If the tetrahedral mesh has 10,000 elements and the polynomial order is $p = 4$ (a fairly coarse discretization), storing the residual vector requires 16.8MB, whereas storing the nonzero blocks in the $\boldsymbol{A}$ matrix requires 17.6GB assuming double precision. While these memory requirements appear to be very high for a problem with only about 2 million degrees of freedom, it should be noted that the calculations required for computing the matrix elements are very costly and scale very unfavorably with large $p$, so a matrix-free solver would likely be prohibitively slow. In either case, it is clear that for practical applications a parallel computer is required even for solving problems that would be considered relatively small for a low order method.

We should also point out that for some methods the off-diagonal blocks are not full, and therefore some savings can be obtained by exploiting the subblock structure. In particular, the $\boldsymbol{A}$ matrix corresponding to the CDG method used here can be implemented with less than half of the storage required if we assume that the off-diagonal blocks are full. For this paper, however, we will not exploit this fact, and we will assume that the nonzero off-diagonal blocks are full.

## 3. Krylov solvers.

We use Krylov subspace methods to solve the linear system $\boldsymbol{Au} = \boldsymbol{b}$. In general, Krylov methods must be preconditioned to perform well. This amounts to finding an approximate solver for $\boldsymbol{Au} = \boldsymbol{b}$ which is relatively inexpensive to apply. From an implementation perspective these methods do not require the explicit form of the system matrix $\boldsymbol{A}$ but only its effect on a given vector $\boldsymbol{p}$, that is, $\boldsymbol{Ap}$. Similarly, the effect of a left preconditioner requires only the ability to approximately calculate $\boldsymbol{A}^{-1}\boldsymbol{p}$.

Our plan is to use a stand-alone multilevel solver as a preconditioner for the Krylov method. This strategy is commonly used [29, 17, 26, 25] and is found to add robustness over a multilevel iterative method at an affordable cost.

We consider several unsymmetric Krylov subspace methods—the quasi-minimal residual method (QMR), the conjugate gradient squared method (CGS), the generalized minimal residual method (GMRES), and restarted GMRES with restart value $r$ (GMRES(r)). Among these, GMRES is, in general, the fastest and most reliable since it is based on the minimization of the true residual, but its storage and computational cost increases with the number of iterations. GMRES(r) is an attempt to resolve this; however, it is well known that its convergence may occasionally stagnate. The two methods CGS and QMR are variants of the biorthogonalization method BiCG and require a lower amount of storage and computation than GMRES. More details on these methods can be found in [2, 38].

## 4. ILU0 preconditioning.

Incomplete matrix factorizations are commonly used as general purpose preconditioners for Krylov solvers. In this section, we describe two preconditioners, the block Jacobi and the block GS methods, which are obtained by setting to zero certain entries in the original matrix, as well as the incomplete LU factorization with zero fill, ILU0. We shall see that the ILU0 method is an attractive option for DG preconditioning, because it offers a superior performance at a comparable cost to the other simpler methods. In addition, by making very weak assumptions on the quality of the meshes, one can make simplifications to the basic ILU0 algorithm to further reduce the cost performance and storage requirements.

For a DG mesh consisting of $ne$ elements, we think of the matrix $\boldsymbol{A}$ as consisting of $ne$-by-$ne$ blocks. Further, we let $\boldsymbol{A}_{ij}$ denote the block with indices $i, j$. The goal is to compute a matrix $\tilde{\boldsymbol{A}}$ which approximates $\boldsymbol{A}$ and which allows for the computation of $\tilde{\boldsymbol{A}}^{-1}\boldsymbol{p}$ for an arbitrary vector $\boldsymbol{p}$ in an inexpensive manner.

**4.1. The block diagonal preconditioner.** A simple preconditioner is obtained by setting all blocks except the ones on the diagonal to zero:

$$\tilde{\boldsymbol{A}}^{\mathrm{J}}_{ij} = \begin{cases} \boldsymbol{A}_{ij} & \text{if } i = j, \\ \boldsymbol{0} & \text{if } i \neq j. \end{cases} \tag{4.1}$$

This block diagonal preconditioner, or *block Jacobi*, is very easy to compute and the effect of its inverse on an arbitrary vector is relatively cheap to compute since all the diagonal blocks are decoupled and can be processed separately. It can have an acceptable performance for some particular problems and parameter choices, but as we will show it can also perform very poorly for more general problems.

If we keep the diagonal blocks plus all the blocks above (or below) the diagonal, we obtain the block GS preconditioner:

$$\tilde{\boldsymbol{A}}^{\mathrm{GS}}_{ij} = \begin{cases} \boldsymbol{A}_{ij} & \text{if } i \leq j, \\ \boldsymbol{0} & \text{if } i > j. \end{cases} \tag{4.2}$$

This matrix is also trivial to obtain from $\boldsymbol{A}$, and although the actual inverse is nontrivial, the effect of its inverse on a vector can be easily calculated by a block backsolve. For some specific problems such as pure convection with an appropriate element numbering, the GS method can perform significantly better than Jacobi. For general problems, however, it can be expected to give only small factors of improvement.

Note that unlike the scalar Jacobi and GS methods, the blocked versions require preprocessing of $\boldsymbol{A}$ for computing the inverses (or the LU factorizations) of the diagonal blocks $\boldsymbol{A}_{ij}$. For the large block sizes obtained when $p$ and $m$ are large, this can become a substantial part of the total preconditioning cost; see section 4.4 for more details. To some extent, this justifies the use of more sophisticated block preconditioners which incur small additional expense.

**4.2. LU factorizations.** We start by considering a full block LU factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ which can be computed by Gaussian elimination. A dense version without pivoting that does not take sparsity into account is

```
$\boldsymbol{U} \leftarrow \boldsymbol{A}, \boldsymbol{L} \leftarrow \boldsymbol{I}$
for $j = 1$ to $n - 1$
    for $i = j + 1$ to $n$
        $\boldsymbol{L}_{ij} = \boldsymbol{U}_{ij}\boldsymbol{U}_{jj}^{-1}$
        for $k = j + 1$ to $n$
            $\boldsymbol{U}_{ik} = \boldsymbol{U}_{ik} - \boldsymbol{L}_{ij}\boldsymbol{U}_{jk}$
        end for
    end for
end for
```

Using this full factorization would solve the problem exactly in one iteration, at least in the absence of rounding errors. Unfortunately, this factorization typically requires a large amount of storage for the *fill-in* and is computationally prohibitive. Here the fill-in refers to the matrix entries that are zero in $\boldsymbol{A}$ but nonzero in either $\boldsymbol{U}$ or $\boldsymbol{L}$.

A compromise is to compute an approximate factorization $\tilde{\boldsymbol{A}}^{ILU} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}} \approx \boldsymbol{A}$, such as the incomplete LU factorization [30]. In the so-called ILU0 method, no matrix entries outside the sparsity pattern of $\boldsymbol{A}$ are allowed into the factorization $\tilde{\boldsymbol{L}}, \tilde{\boldsymbol{U}}$, and any such entries are ignored during the factorization. This factorization is cheap to compute and requires about the same memory storage as $\boldsymbol{A}$ itself. Although the performance of the method is hard to analyze, the resulting matrix $\tilde{\boldsymbol{A}}^{\mathrm{ILU}}$ is often an efficient preconditioner. More expensive alternatives that perform better typically discard most of the fill-in but not all. For example, some methods keep the fill-in based on a prescribed pattern, whereas some other approaches discard matrix entries which are smaller than a prescribed threshold [38].

In the block matrices that arise from our DG discretizations, the sparsity pattern of $\boldsymbol{A}$ is given directly by the connectivity between the mesh elements. That is, $\boldsymbol{A}_{ij} \neq \boldsymbol{0}$ only if $i = j$ or elements $i, j$ are neighbors. The ILU0 algorithm can then be written as follows:

```
$\tilde{\boldsymbol{U}} \leftarrow \boldsymbol{A}, \tilde{\boldsymbol{L}} \leftarrow \boldsymbol{I}$
**for** $j = 1$ **to** $n - 1$
    **for** neighbors $i > j$ of element $j$
        $\tilde{\boldsymbol{L}}_{ij} = \tilde{\boldsymbol{U}}_{ij}\tilde{\boldsymbol{U}}_{jj}^{-1}$
        $\tilde{\boldsymbol{U}}_{ii} \leftarrow \tilde{\boldsymbol{U}}_{ii} - \tilde{\boldsymbol{L}}_{ik}\tilde{\boldsymbol{U}}_{ki}$
        **for** neighbors $k > j$ of elements $j$ and $i$
            $\tilde{\boldsymbol{U}}_{ik} = \tilde{\boldsymbol{U}}_{ik} - \tilde{\boldsymbol{L}}_{ij}\tilde{\boldsymbol{U}}_{jk}$
        **end for**
    **end for**
**end for**
```

We can simplify this ILU0 algorithm by noting that for our meshes, it is uncommon that an element $k$ is a neighbor of both elements $j$ and $i$ when $j$ is a neighbor of $i$. For a two dimensional triangular mesh, this would imply that the three triangles $i, j, k$ are fully connected, which means that the mesh is of rather poor quality. When calculating the ILU0 preconditioner, we assume that this situation never occurs. If it does, we just ignore the connection between $i$ and $k$ since after all we are just computing an approximate factorization. Our final ILU0 algorithm then gets the simple form

```
$\tilde{\boldsymbol{U}} \leftarrow \boldsymbol{A}, \tilde{\boldsymbol{L}} \leftarrow \boldsymbol{I}$
**for** $j = 1$ **to** $n - 1$
    **for** neighbors $i > j$ of $j$
        $\tilde{\boldsymbol{L}}_{ij} = \tilde{\boldsymbol{U}}_{ij}\tilde{\boldsymbol{U}}_{jj}^{-1}$
        $\tilde{\boldsymbol{U}}_{ii} \leftarrow \tilde{\boldsymbol{U}}_{ii} - \tilde{\boldsymbol{L}}_{ik}\tilde{\boldsymbol{U}}_{ki}$
    **end for**
**end for**
```

We note that this simplification gave our factorization another attractive property, namely, that $\tilde{\boldsymbol{U}}_{ij} = \boldsymbol{A}_{ij}$ when $j > i$. In other words, $\tilde{\boldsymbol{U}}$ differs from $\boldsymbol{A}$ only in the diagonal blocks, which means that we do not have to store the upper triangular blocks of $\tilde{\boldsymbol{U}}$. Another possibility for reducing storage costs is to utilize the fact that for a true ILU0 factorization, $\boldsymbol{A}_{ij} = \tilde{\boldsymbol{A}}_{ij}^{\mathrm{ILU}} = (\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}})_{ij}$ when $\boldsymbol{A}_{ij}$ is nonzero [38]. That is, $\tilde{\boldsymbol{A}}^{\mathrm{ILU}}$ differs from $\boldsymbol{A}$ only outside the sparsity pattern of $\boldsymbol{A}$. Therefore, the matrix $\boldsymbol{A}$ can be overwritten by $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{U}}$, and any operations involving $\boldsymbol{A}$ can be performed using $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{U}}$. For efficiency reasons one would likely also precompute and store the LU factors of the diagonal blocks of $\tilde{\boldsymbol{U}}$. We observe that, while this approach reduces the total storage requirements, the matrix $\boldsymbol{A}$ is not stored explicitly, and therefore, its application to an arbitrary vector incurs a small computational penalty. In our implementation we take advantage of the first property, namely, we store $\boldsymbol{A}$, the lower triangular blocks of $\tilde{\boldsymbol{L}}$, but only the diagonal blocks of $\tilde{\boldsymbol{U}}$.

**4.3. Minimum discarded fill ordering.** The performance of an incomplete LU factorization when used as a preconditioner might be highly dependent on the order in which the elimination is performed (or, equivalently, the ordering of the unknowns assuming the elimination is done from top to bottom). The issue of ordering for incomplete LU factorizations has been studied before mostly for diffusion dominated flows. To our knowledge, however, this has not been done in the context of DG discretizations.

It has been reported that a reverse Cuthill–McKee (RCM) ordering [15] performs systematically well for certain classes of problems [4]. Generally, however, this is not the case for more complex problems involving high convection and anisotropic elements. Various physically inspired orderings have been suggested. For example, in [24], it is shown that by ordering the elements along each of the streamlines of the convective operator good performance is obtained for the GS method. However, these approaches do not generalize well to arbitrary problems and do not provide intuition on how to treat mesh anisotropy and nonconvective terms.

A more general approach is the minimum discarded fill (MDF) method [10], where the element that produces the least discarded fill-in is eliminated first, and this process is repeated in a greedy way. The algorithm is similar to the minimum degree algorithm for fill-reduction in exact factorizations [16], but it tries to minimize the value of the ignored fill rather than the size of the fill-in. The MDF method was reported to give good results, mostly for scalar elliptic problems, although the procedure for computing it was expensive. An alternative ordering method which is cheaper to evaluate but provides comparable performance was recently proposed in [28].

For our systems, it is clear that we would like to retain the block format of the linear system and will therefore consider the ordering of the elements and not the individual unknowns. Our algorithm is similar to the MDF method; however, it is adapted to our block matrices with appropriate simplifications to reduce its cost.

Consider step $j$ of the ILU algorithm, when $j-1$ elements have already been eliminated. The original algorithm would proceed by using element $j$ as the pivot element. Here, we instead compute the fill $\Delta\tilde{\boldsymbol{U}}^{(j)}$ that would be generated if element $j'$ was chosen as the pivot element:

$$\Delta\tilde{\boldsymbol{U}}_{ik}^{(j,j')} = -\tilde{\boldsymbol{U}}_{ij'}\tilde{\boldsymbol{U}}_{j'j'}^{-1}\tilde{\boldsymbol{U}}_{j'k} \qquad \text{for neighbors } i \ge j, k \ge j \text{ of element } j', \tag{4.3}$$

for all potential pivots $j' \ge j$. The matrices $\Delta\tilde{\boldsymbol{U}}^{(j,j')}$ are the errors that we introduce at step $j$ of the incomplete LU algorithm compared to an exact LU step. To measure the size of $\tilde{\boldsymbol{U}}^{(j,j')}$ we take the Frobenius matrix norm and assign the weight

$$w^{(j,j')} = \|\Delta\tilde{\boldsymbol{U}}^{(j,j')}\|_{\mathrm{F}}. \tag{4.4}$$

Clearly, a greedy way for choosing a good pivot element at step $j$ is to pick the one that minimizes $w^{(j,j')}$. One can then proceed by switching rows $j$ and $j'$, performing the ILU elimination step, and continuing to step $j+1$, where the procedure is repeated. This is the basis of our algorithm.

In order to make the algorithm faster and easier to implement, we will make some simplifications that do not appear to destroy the nice properties of the ordering. To begin with, it seems unnecessary to compute the full product in (4.3) since we are concerned only about its matrix norm. Inspired by the fact that

$$\|\Delta\tilde{\boldsymbol{U}}_{ik}^{(j,j')}\|_{\mathrm{F}} = \| -\tilde{\boldsymbol{U}}_{ij'}\tilde{\boldsymbol{U}}_{j'j'}^{-1}\tilde{\boldsymbol{U}}_{j'k}\|_{\mathrm{F}} \le \|\tilde{\boldsymbol{U}}_{ij'}\|_{\mathrm{F}}\|\tilde{\boldsymbol{U}}_{j'j'}^{-1}\tilde{\boldsymbol{U}}_{j'k}\|_{\mathrm{F}}, \tag{4.5}$$

we premultiply the matrix by its block diagonal inverse and reduce each block $\boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{ij}$ to the scalar number $\|\boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{ij}\|_{\mathrm{F}}$. This implies a considerable reduction in size and computational requirements since only the scalar fill-in weights, rather than the full blocks, are needed in the ordering algorithm.

We summarize our ILU0 ordering algorithm as follows. The input to the algorithm is the matrix $\boldsymbol{A}$, and the output is the computed ordering (permutation) $p_i$.

```
$\boldsymbol{B} \leftarrow (\tilde{\boldsymbol{A}}^{\mathrm{J}})^{-1}\boldsymbol{A}$            *Premultiply by block diagonal*
$C_{ij} \leftarrow \|\boldsymbol{B}_{ij}\|_{\mathrm{F}}$                 *Reduce each block to scalar*
**for** $k = 1, \dots, n$                    *Compute all weights*
    $\Delta C \leftarrow 0$
    **for** neighbors $i, j$ of element $k$, $i \neq j$,
        $\Delta C_{ij} \leftarrow C_{ik}C_{kj}$              *Discarded fill matrix*
    **end for**
    $w_k \leftarrow \|\Delta C\|_{\mathrm{F}}$                 *Fill weight*
**end for**
**for** $i = 1, \dots, n$                    *Main loop*
    $p_i \leftarrow \mathrm{argmin}_j w_j$              *Pivot with smallest fill-in*
    $w_{p_i} \leftarrow \infty$                    *Do not choose $p_i$ again*
    **for** neighbors $k$ of $p_i$ not yet numbered   *Update weights*
        Recompute $w_k$, considering only
        neighbors not yet numbered
    **end for**
**end for**
```

The weights $w_j$ are stored in a min-heap data structure to obtain a total running time of $\mathcal{O}(n \log n)$ for the algorithm. The majority of the computational cost is typically the initial multiplication by the block diagonal inverse matrix, except possibly for very low $p$. We have experimented with cheaper versions of the algorithm, such as projecting all blocks to $p = 0$ before multiplying by the block diagonal. This produced good results considering the simplicity; however, in our implementation we work with the full block diagonal as in the algorithm above.

The orderings produced by the above algorithm are usually not appropriate for the simpler GS method. However, we can make a small modification to find good ordering for certain problems (in particular, purely convective problems). At each step, we choose the element that gives the smallest error between the true matrix and the block upper triangular GS matrix $\tilde{\boldsymbol{A}}^{\mathrm{GS}}$. This is accomplished by using the above ILU0 ordering algorithm but modifying the computation of the weights $\Delta C_{ij}$ according to

```
...
$\Delta C \leftarrow 0$
**for** neighbors $i$ of element $k$
    $\Delta C_{ij} \leftarrow C_{ik}$                    *Discarded lower triangular blocks*
**end for**
...
```

The algorithm will then try to find a permutation such that the matrix becomes upper triangular, in which case the GS approximation is exact (as well as the ILU0 approximation).

We note that for a pure upwinded convective problem, the MDF ordering is optimal since at each step it picks an element that either does not affect any other elements (downwind) or does not depend on any other element (upwind), resulting in a perfect factorization. But the algorithm works well for other problems, too, including convection-diffusion and multivariate problems, since it tries to minimize the error between the exact and the computed LU factorizations. It also takes into account the effect of the discretization (e.g., highly anisotropic elements) on the ordering—something that is harder to do with ad hoc methods based on physical observations.

**4.4. Computational cost.** To quantify the relative computational cost of the three preconditioners, we calculate the leading terms in the operation count for each algorithm. Consider a discretization with block size $N$, dimension $D$, and number of elements $ne$. All block operations are then performed on dense $N$-by-$N$ matrices and $N$-by-1 vectors, and each block row of the matrix $\boldsymbol{A}$ consists of a diagonal block plus $D+1$ off-diagonal blocks. The number of floating point operations (flops) for basic dense linear algebra operations are

- $N^2$ flops for a triangular back-solve,
- $2N^2$ flops for a matrix-vector product,
- $2N^3$ flops for a matrix-matrix product, and
- $(2/3)N^3$ flops for an LU factorization.

Using this we obtain the following costs for the precalculation, that is, operations performed only once for each matrix $\boldsymbol{A}$:

| Operation | Flop count |
|---|---|
| Jacobi factorization $\tilde{\boldsymbol{A}}^J$ | $(2/3)N^3ne$ |
| GS factorization $\tilde{\boldsymbol{A}}^{GS}$ | $(2/3)N^3ne$ |
| ILU(0) factorization $\tilde{\boldsymbol{A}}^{ILU}$ | $(2D+8/3)N^3ne$ |

Note that the precalculation also includes the computation of the actual matrix $\boldsymbol{A}$. The cost of this is application and implementation dependent, but for our nonlinear problems it scales as $\mathcal{O}(N^3ne)$ with a relatively large constant. This means that even though the precalculation step for ILU(0) is about a magnitude more expensive than for Jacobi or GS, the ratio of the total cost is typically much smaller. We also point out that for simplicity we consider only full $N$-by-$N$ blocks, but if the off-diagonal blocks are sparser than the diagonal blocks, such as in the CDG scheme that we are using [34], the cost of an ILU(0) factorization drops significantly.

For the operations performed during the GMRES iterations we obtain the following costs per iteration:

| Operation | Flop count |
|---|---|
| Jacobi solve $(\tilde{\boldsymbol{A}}^{J})^{-1}p$ | $(2)N^2ne$ |
| GS solve $(\tilde{\boldsymbol{A}}^{GS})^{-1}p$ | $(D+3)N^2ne$ |
| ILU(0) solve $(\tilde{\boldsymbol{A}}^{ILU})^{-1}p$ | $(2D+4)N^2ne$ |
| Matrix-vector product $\boldsymbol{Ax}$ | $(2D+4)N^2ne$ |

The cost of a matrix-vector product is the same as a solve with the ILU(0) preconditioner. Therefore, since every iteration includes one matrix-vector product and one preconditioner solve, the difference per iteration between Jacobi and ILU(0) is always less than a factor of two.

For the MDF element ordering algorithm, the cost is dominated by the first line (block scaling) which has the same cost as computing an ILU(0) factorization. However, in practice we do not perform this operation once for every linear solve but typically only once per timestep or even once per problem. Therefore, the ordering does not contribute significantly to the total computational cost.

## 5. Coarse scale corrections.

Multilevel methods, such as multigrid [19] solve the system $\boldsymbol{Au} = \boldsymbol{b}$ by introducing coarser level discretizations. This coarser discretization can be obtained either by using a coarser mesh ($h$-multigrid) or, for high order methods, by reducing the polynomial degree $p$ ($p$-multigrid [37, 14]). The residual is restricted to the coarse scale where an approximate error is computed, which is then applied as a correction to the fine scale solution. A few iterations of a smoother (such as Jacobi's method) are applied after and possibly before the correction to reduce the high frequency errors.

In the multigrid method a hierarchy of levels is used which are traversed in a recursive way with a few smoothing iterations performed at each level. At the coarsest scale, the problem is usually solved exactly, either by a different technique (such as a direct solver) or by iteration. The multigrid method can be shown theoretically to achieve optimal asymptotic computational cost, at least for elliptic problems.

For our DG discretizations, it is natural and practical to consider coarser scales obtained by reducing the polynomial order $p$. The problem size is highly reduced by decreasing the polynomial order to $p = 0$ or $p = 1$, even from moderately high orders such as $p = 4$. For very large problems it may be necessary to consider $h$-multigrid approaches to solve the coarse grid problem. However, this is not considered in this paper.

Furthermore, we have observed that we often get better overall performance by using a simple two-level scheme where the fine level corresponds to $p = 4$ and the coarse level is either $p = 1$ or $p = 0$ rather than a hierarchy of levels. This can be explained partly by the following arguments: (i) The block Jacobi smoother already solves the problem exactly within each block; in addition, the block ILU0 accounts for some of the interelement connectivities. The main role of the coarse scale correction is therefore to account for the connectivities between the elements not handled by the smoother. (ii) The cost of computing the intermediate levels and the corresponding smoothers using the DG method is high, even for degrees as low as $p = 2$. Therefore, we appear to be better off making a cheap single coarse scale correction and spending more of the time in the fine scale smoother and the GMRES iterations.

Based on these observations, our preconditioning algorithm becomes very simple as it considers only two levels. It solves the linear system $\boldsymbol{Au} = \boldsymbol{b}$ approximately using a single coarse scale correction:

```
0. $\boldsymbol{A}^{(0)} = \boldsymbol{P}^T\boldsymbol{A}\boldsymbol{P}$        Precompute coarse operator blockwise
1. $\boldsymbol{b}^{(0)} = \boldsymbol{P}^T\boldsymbol{b}$          Restrict residual element/componentwise
2. $\boldsymbol{A}^{(0)}\boldsymbol{u}^{(0)} = \boldsymbol{b}^{(0)}$       Solve coarse scale problem
3. $\boldsymbol{u} = \boldsymbol{P}\boldsymbol{u}^{(0)}$            Prolongate solution element/componentwise
4. $\boldsymbol{u} = \boldsymbol{u} + \alpha\tilde{\boldsymbol{A}}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{u})$  Apply smoother $\tilde{\boldsymbol{A}}$ with damping $\alpha$
```

Commonly used smoothers $\tilde{\boldsymbol{A}}$ include block Jacobi $\tilde{\boldsymbol{A}}^{\mathrm{J}}$ or GS $\tilde{\boldsymbol{A}}^{\mathrm{GS}}$, but as we describe below, we are getting large performance gains by using the ILU0 factorization $\tilde{\boldsymbol{A}}^{\mathrm{ILU}}$. The restriction/prolongation operator $\boldsymbol{P}$ is a block diagonal matrix with all the blocks identical. The prolongation operator has the effect of transforming the solution from a nodal basis to a hierarchical orthogonal basis function based on Koornwinder polynomials [27] and setting the coefficients corresponding to the higher modes equal to zero. The transpose of this operator is used for the restriction of the weighted residual and for the projection of the matrix blocks. For more details on these operators we refer the reader to [7].

We use a smoothing factor $\alpha = 2/3$ when the block diagonal smoother is used to ensure that it has good smoothing properties (which reduces the error in the high frequencies), while $\alpha = 1$ is used for the GS and ILU0 smoothers. We use a direct sparse solve for the linear system in step 2, and we note that $\boldsymbol{A}^{(0)}$ is usually magnitudes smaller than $\boldsymbol{A}$.

### 5.1. The ILU0/coarse scale preconditioner.

It is well known that an ILU0 factorization can be used as a smoother for multigrid methods [41, 42], and it has been reported that it performs well for the Navier–Stokes equations, at least in the incompressible case using low order discretizations [12]. Inspired by this, we use the block ILU0 factorization $\tilde{\boldsymbol{A}}^{\mathrm{ILU}}$ as a postsmoother for a two-level scheme.

Our numerical experiments indicate that the block ILU0 preconditioner and the low degree correction preconditioner complement each other. With our MDF element ordering algorithm, the ILU0 performs almost optimally for highly convective problems, while the coarse scale correction with block diagonal, block GS, or block ILU0 postsmoothing performs very well in the diffusive limit.

## 6. Results.

### 6.1. Test procedure.

To study the performance of the iterative solvers, we consider two equations, the linear scalar convection-diffusion equation and the compressible Navier–Stokes equations. For the Navier–Stokes equations, the linear problem is obtained by linearizing the governing equations about a steady-state solution. We note that in a practical nonlinear solver, this linearized problem is different in each Newton step and in each new timestep. However, in order to make a systematic study possible, we choose the representative steady-state solution for all our linearizations. This leads to a problem of the form $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{M} - \Delta t\boldsymbol{K}$, where the right-hand side $\boldsymbol{b}$ is a random vector, which is taken to be the same for all cases considered. To investigate the solution behavior for very large timesteps (i.e., steady state), we set $\Delta t = \infty$, and we take $\boldsymbol{A} = \boldsymbol{K}$.

Our initial solution vector is always the zero vector, and during the iterative process, we monitor the norm of the true error $\boldsymbol{e} = \boldsymbol{u} - \boldsymbol{A}^{-1}\boldsymbol{b}$ relative to the initial error (which is minus the true solution). We iterate until a relative error of $10^{-3}$ is obtained and study the number of iterations required by the various methods. Note that the residual in the iterative solver depends on the preconditioner, and therefore, its magnitude is not an appropriate convergence criterion.

Table 6.1
*The preconditioners that we consider for our test problems.*

| Preconditioner | Description |
|---|---|
| BJ | block Jacobi $\tilde{\boldsymbol{A}}^{\mathrm{J}}$ |
| BGS | block GS $\tilde{\boldsymbol{A}}^{\mathrm{GS}}$ |
| BILU0 | block ILU(0) factorization $\tilde{\boldsymbol{A}}^{\mathrm{ILU}}$ |
| BJ-p# | $p = \#$ correction, smoother $\tilde{\boldsymbol{A}}^{\mathrm{J}}$, $\alpha = 2/3$ |
| BGS-p# | $p = \#$ correction, smoother $\tilde{\boldsymbol{A}}^{\mathrm{GS}}$, $\alpha = 1$ |
| BILU0-p# | $p = \#$ correction, smoother $\tilde{\boldsymbol{A}}^{\mathrm{ILU}}$, $\alpha = 1$ |

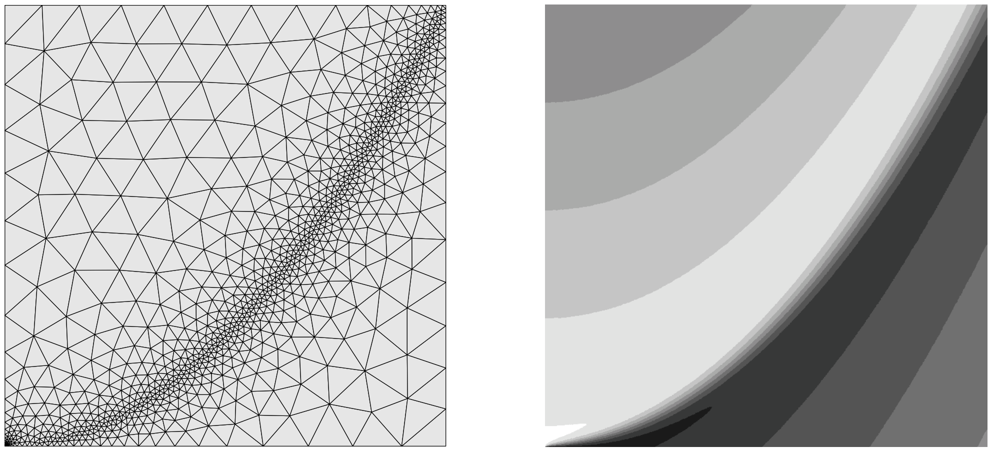

(a) Mesh (b) Solution ($\varepsilon = 10^{-2}$)

Fig. 6.1. *The convection-diffusion model problem.*

The preconditioners that we consider are listed in Table 6.1. We do not consider the actual computational time for these cases but only the number of iterations required for convergence. The operations counts in section 4.4 give good estimates of the relative cost between the preconditioners as well as the precalculation steps such as the MDF ordering and the ILU factorizations.

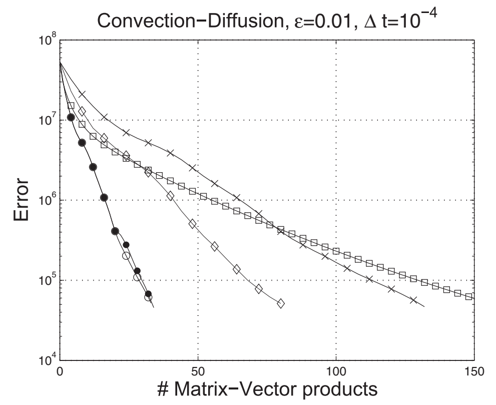
**6.2. Scalar convection-diffusion.** Our first test case is a scalar convection-diffusion problem in two dimensions of the form

$$\frac{\partial u}{\partial t} + \nabla \cdot \begin{bmatrix} \alpha u \\ \beta u \end{bmatrix} - \nabla \cdot \begin{bmatrix} \varepsilon u_x \\ \varepsilon u_y \end{bmatrix} = 0 \tag{6.1}$$

with the space-dependent divergence-free convection field $(\alpha, \beta) = (1, 2x)$ and diffusion $\varepsilon \geq 0$. We solve on the unit square with boundary conditions $u = x - 1$ at the bottom edge, $u = 1 - y$ at the left edge, and free boundary conditions on the upper and right edges.

We use a highly graded isotropic triangular mesh (see Figure 6.1(a)) with 1961 elements and fourth order polynomials within each element ($p = 4$). A typical solution $u(x, y)$ for parameter value $\varepsilon = 10^{-2}$ is shown in Figure 6.1(b).

**6.2.1. Iterative solvers.** First, we solve our test problem with the block Jacobi method as well as four different Krylov subspace solvers, QMR, CGS, GMRES(20), and GMRES, all preconditioned using the block diagonal preconditioner. The results are shown in Figure 6.2 for $\varepsilon = 0.01$ and $\Delta t = 10^{-4}$ and $10^{-2}$, corresponding to



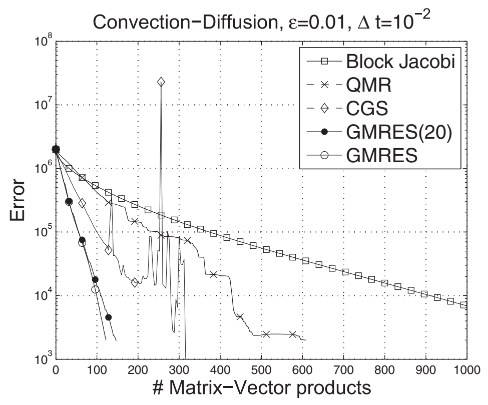


FIG. 6.2. *Convergence of various iterative solvers for the convection-diffusion problem with* $\varepsilon = 0.01$ *and timestep* $\Delta t = 10^{-4}$*, CFL* $= 3{,}000$ *(left), and* $\Delta t = 10^{-2}$*, CFL* $= 300{,}000$ *(right).*

CFL numbers of about $3,000$ and $300,000$, respectively. We note that for the smaller timestep, all solvers show uniform convergence, with CGS requiring about twice as many matrix-vector products as GMRES and block Jacobi and QMR about three times as many. The situation is similar for the larger timestep; however, CGS and QMR are more erratic than GMRES and block Jacobi performs relatively worse. This becomes more extreme for larger timesteps or for more complex problems, where Jacobi in general does not converge at all. The plots also show that the restarted GMRES gives only a slightly slower convergence than the more expensive full GMRES for this problem.

The full GMRES has the disadvantage that the cost of storage and computation increases with the number of iterations. We have not observed any stagnation of the restarted GMRES(20) for our problems, and its slightly slower convergence is well compensated by the lower cost of the method. The solvers QMR and CGS could be good alternatives since they are inexpensive and converge relatively fast. Based on these observations and the fact that similar results are obtained for other problem types, we focus on the GMRES(20) solver in the remainder of the paper.

#### 6.2.2. Element orderings for ILU0 and GS.

Next we consider the steady-state solution $\Delta t = \infty$ for varying $\varepsilon$ and solve using the GMRES(20) solver with ILU0 and GS preconditioning. The elements are ordered using two different methods: (i) reverse Cuthill–McKee (RCM), which orders the elements in a breadth-first fashion attempting to minimize the bandwidth of the factorized matrix and which has been suggested as a good ordering for incomplete LU factorizations [4], and (ii) minimum discarded fill (MDF) with our simplifications as described in section 4 for ILU0 and for GS.

The number of GMRES iterations required for convergence is shown in Figure 6.3 as a function of $\varepsilon$. We note that for diffusion dominated problems (large $\varepsilon$) the results are relatively independent of the ordering, while for convection dominated problems (small $\varepsilon$) the ordering has a very large impact. In particular, ILU0 with the MDF ordering is two orders of magnitude more efficient than RCM for low $\varepsilon$ and converges in an almost optimal way with only two iterations for $\varepsilon = 10^{-6}$. This is because the algorithm will number the elements such that very little fill is produced, giving an ILU0 factorization that is close to the true LU factorization.

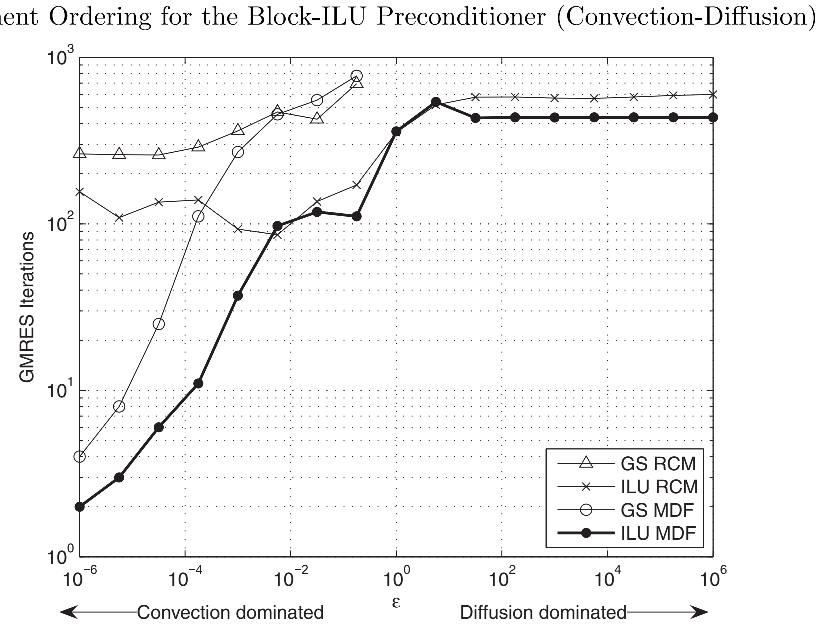


FIG. 6.3. *The effect of element ordering on the block ILU0 and the GS preconditioners for the convection-diffusion problem. Our MDF method is almost perfect for convection dominated problems, but for higher diffusion the effect of element ordering is smaller.*

The GS preconditioner also performs almost optimally for the highly convective problem, again because the elements can be ordered so that the system is almost upper triangular, making the GS approximation exact. However, even for very small amounts of diffusion added to the system this convergence is destroyed, and the number of iterations increases faster than for ILU0.

Based on these results, we will use the MDF ordering for all remaining tests.

**6.2.3. Comparison of preconditioners.** Next we study the performance of the different preconditioners as a function of $\varepsilon$, again for the steady-state case $\Delta t = \infty$. In Figure 6.4 we plot the number of iterations required, and we can draw the following conclusions:

- The BJ preconditioner performs very poorly in all problem regimes.
- The BILU0 preconditioner performs almost perfectly for convection dominated problems but poorly for diffusion dominated problems.
- The BJ-p1 preconditioner is approximately the complement of ILU, since it performs very well for the diffusive problems but poorly for convective problems.
- The BILU0-p1 preconditioner combines the best of the BILU0 and the BJ-1 preconditioners and is consistently better than any of the two for all combinations of convection and diffusion problems.
- The BGS-p1 preconditioner performs similarly to BILU0-p1 for this example. This is consistent with the results reported in [24]. However, as our next examples show, this is no longer the case for more complex problems.

The above results suggest the BILU0-p1 preconditioner, namely, trying to combine the strengths of the BILU0 preconditioner and the $p = 1$ correction in a general purpose method. Furthermore, as we will show in the next section, it performs very well for the Navier–Stokes equations.

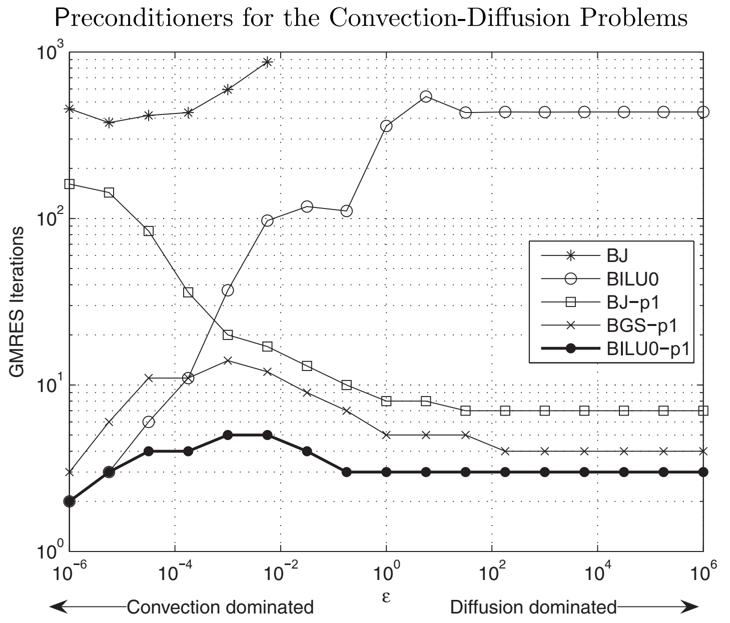


FIG. 6.4. *The number of iterations required for convergence using five preconditioners as a function of* $\varepsilon$*. The BILU*0 *and BJ-p*1 *preconditioners perform well in different problem regimes, and the BILU0-p*1 *preconditioner combines the two into an efficient general purpose method. For this simple case the BGS-p*1 *also performs close to optimally.*

**6.2.4. Timestep dependence.** To study the convergence for the convection-diffusion problem as a function of the timestep $\Delta t$, we consider the two limiting regimes—pure convection ($\varepsilon = 0$) and pure diffusion ($\varepsilon = \infty$, which reduces to Poisson's equation). We plot the number of iterations required for convergence as a function of $\Delta t$ using four preconditioners. The dashed lines in the plots correspond to the explicit limit, that is, the number of timesteps an explicit solver would need to reach the corresponding time $\Delta t$. An implicit solver should be significantly below these lines in order to be efficient. Note that this comparison is not very precise, since the overall expense depends on the relative cost between residual evaluation and matrix-vector products/preconditioning. Nevertheless, it provides some useful intuition.

The results are shown in Figure 6.5. As before, the ILU0-based preconditioners are essentially perfect for the pure convection problem (convergence in one or two iterations) independently of the timestep. The BJ and the BJ-p0 preconditioners have about the same convergence, which means that the coarse scale correction does not improve the convergence at all, which is expected for purely convective problems.

For the diffusive problem the BJ-p0 preconditioner is almost as good as the BILU0-p0 preconditioner for large $\Delta t$. For smaller timesteps the ILU0 is somewhat better than the BJ-p0, and as before, the BILU0-p0 is always as good as or better than any other preconditioner.

**6.2.5. Scaling with *h* and *p*.** Our final test for the convection-diffusion problem is to study the scaling of the convergence rate as the mesh is refined, both by refining the elements ($h$-refinement) and by increasing the polynomial order ($p$-refinement). We consider the same convection-diffusion problem (6.1) as before, but using a regular

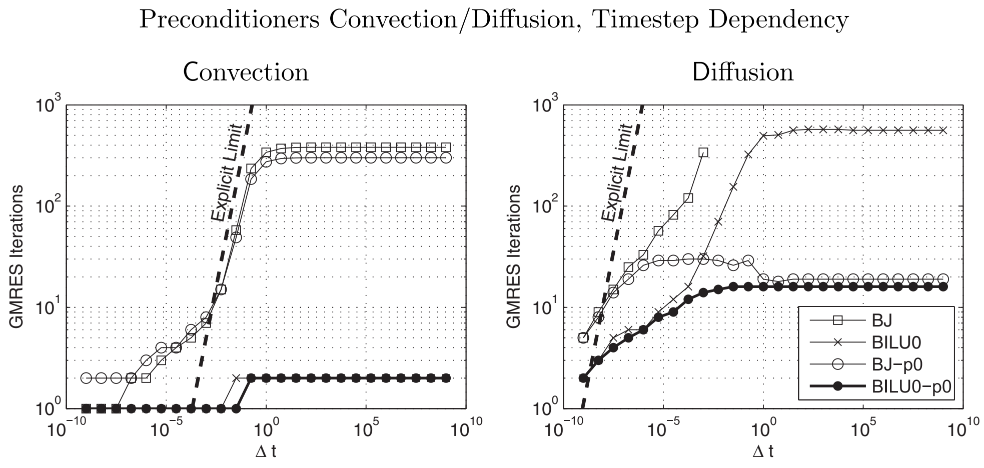


FIG. 6.5. *The convergence of four preconditioners for the pure convection and the pure diffusion problems as a function of the timestep $\Delta t$. The explicit limit line indicates which CFL number each $\Delta t$ corresponds to.*

grid of $n$-by-$n$ squares with edge lengths $h = 1/n$, split into a total of $2n^2$ triangles. The problem is solved using the two preconditioners BILU0-p0 and BILU0-p1, the steady-state timestep $\Delta t = \infty$, and the three diffusion values $\varepsilon = 0$, $\varepsilon = 10^{-3}$, and $\varepsilon = \infty$.

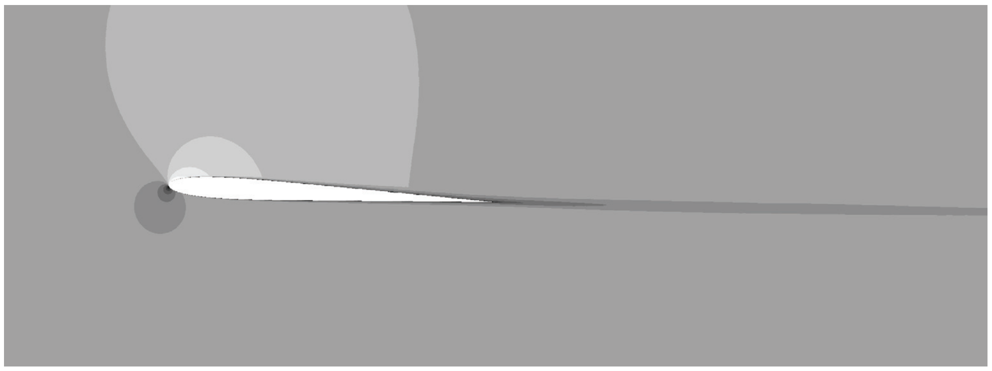
The resulting numbers of iterations are shown in Table 6.2. We note that for $\varepsilon = 0$ (pure convection), both preconditioners are perfect for any values of $h$ and $p$ due to the MDF element ordering. For the low diffusion $\varepsilon = 10^{-3}$, there is essentially no $p$-dependency but a slight $h$-dependence in both preconditioners (very weak for BILU0-p1). Finally, for the Poisson problem $\varepsilon = \infty$, the convergence with BILU0-p1 appears to be independent of both $h$ and $p$, but with BILU0-p0 the number of iterations grows with decreasing $h$ or increasing $p$.

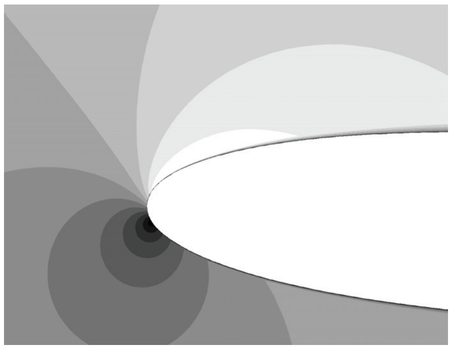
We note that even though the number of iterations is independent of the mesh size $h$ (such as for Poisson with BILU0-p1), this does not necessarily mean that the actual algorithm has optimal convergence, since the size of the coarse scale problem varies. This could of course be remedied by combining the preconditioner with an $h$-multigrid step, assuming a hierarchy of unstructured grids is available. However, in all our experiments with $p = 4$, the coarse scale problems are so much smaller that the computations spent on the coarse scale are negligible. This means that in practice we observe the optimal linear dependence on the number of degrees of freedom when the number of iterations is constant.

**6.3. Compressible Navier–Stokes.** Next, we study a more complex and realistic problem, the solution of compressible Navier–Stokes equations. We will consider a wide range of regimes, including inviscid flow (Euler equations), laminar flow at moderate Reynolds numbers, and high-speed flows using the Reynolds averaged Navier–Stokes (RANS) equations with the one-equation Spalart–Allmaras turbulence model. Our test problem is the flow around an HT13 airfoil at an angle of attack of 3 degrees. A sample solution for the RANS equations and a highly graded structured C-type mesh of 1024 elements and polynomial order $p = 4$ is shown in Figure 6.6. For all the other computations a coarse mesh consisting of 672 elements and polynomial order $p = 4$ within each element is used.

TABLE 6.2
*Scaling of the convergence for the convection-diffusion problem as a function of mesh size $h$ and polynomial order $p$.*

| | | Preconditioner/Iterations | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | BILU0-p0 | | | | | BILU0-p1 | | | | |
| | | $h$=1/2 | $h$=1/4 | $h$=1/8 | $h$=1/16 | $h$=1/32 | $h$=1/2 | $h$=1/4 | $h$=1/8 | $h$=1/16 | $h$=1/32 |
| $\varepsilon = 0$ | $p = 2$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $p = 3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $p = 4$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $p = 5$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $\varepsilon = 10^{-3}$ | $p = 2$ | 2 | 3 | 4 | 5 | 8 | 2 | 3 | 4 | 4 | 4 |
| | $p = 3$ | 2 | 3 | 4 | 5 | 9 | 2 | 3 | 3 | 4 | 4 |
| | $p = 4$ | 2 | 3 | 4 | 6 | 9 | 2 | 3 | 4 | 4 | 5 |
| | $p = 5$ | 2 | 3 | 4 | 6 | 9 | 2 | 3 | 4 | 4 | 5 |
| $\varepsilon = \infty$ | $p = 2$ | 4 | 7 | 10 | 12 | 13 | 2 | 3 | 3 | 3 | 3 |
| | $p = 3$ | 4 | 8 | 11 | 15 | 17 | 3 | 3 | 3 | 3 | 3 |
| | $p = 4$ | 4 | 8 | 15 | 18 | 18 | 3 | 4 | 4 | 3 | 2 |
| | $p = 5$ | 4 | 10 | 17 | 24 | 21 | 3 | 4 | 4 | 4 | 2 |


(a) Solution (Mach number)


(b) Zoom-in, Solution (Mach number)

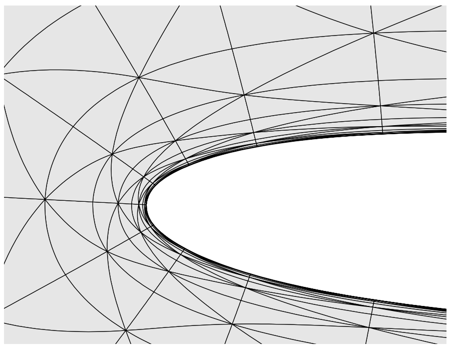

(c) Zoom-in, Mesh

FIG. 6.6. *The compressible Navier–Stokes model problem, with RANS turbulence modeling for $Re = 10^6$ and $M = 0.2$.*

TABLE 6.3
*Convergence of the compressible Navier–Stokes test problem using the four preconditioners and varying Reynolds number, Mach number, and timesteps. A × in the GMRES iterations column indicates that the method did not converge to a relative error norm of $10^{-3}$ in less than 1000 iterations.*

| Problem | Parameters | | Preconditioner/Iterations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Block Jacobi | | | Block GS | | | Block ILU0 | | |
| | $\Delta t$ | $M$ | BJ | BJ-p0 | BJ-p1 | BGS | BGS-p0 | BGS-p1 | BILU0 | BILU0-p0 | BILU0-p1 |
| Inviscid | $10^{-3}$ | 0.2 | 24 | 23 | 19 | 14 | 13 | 11 | 5 | 5 | 4 |
| | $10^{-1}$ | 0.2 | 187 | 125 | 85 | 73 | 55 | 49 | 12 | 9 | 6 |
| | $\infty$ | 0.2 | 840 | 180 | 142 | 456 | 94 | 72 | 40 | 13 | 9 |
| | $10^{-3}$ | 0.01 | 200 | 138 | 112 | 111 | 78 | 67 | 15 | 9 | 6 |
| | $10^{-1}$ | 0.01 | × | × | × | × | × | 532 | 94 | 26 | 10 |
| | $\infty$ | 0.01 | × | × | × | × | × | × | 374 | 50 | 16 |
| Laminar Re = 1,000 | $10^{-3}$ | 0.2 | 50 | 43 | 34 | 25 | 23 | 19 | 4 | 4 | 4 |
| | $10^{-1}$ | 0.2 | × | 867 | 597 | 477 | 378 | 225 | 11 | 8 | 5 |
| | $\infty$ | 0.2 | × | × | × | × | × | × | 37 | 19 | 7 |
| | $10^{-3}$ | 0.01 | 98 | 78 | 66 | 51 | 38 | 33 | 8 | 7 | 5 |
| | $10^{-1}$ | 0.01 | × | 800 | 619 | × | 327 | 207 | 27 | 15 | 9 |
| | $\infty$ | 0.01 | × | × | × | × | × | 748 | 135 | 48 | 12 |
| Laminar Re = 20,000 | $10^{-3}$ | 0.2 | 26 | 25 | 19 | 14 | 13 | 11 | 4 | 4 | 4 |
| | $10^{-1}$ | 0.2 | 456 | 326 | 220 | 219 | 168 | 113 | 16 | 14 | 8 |
| | $\infty$ | 0.2 | × | × | × | × | × | × | 236 | 55 | 20 |
| | $10^{-3}$ | 0.01 | 160 | 117 | 90 | 61 | 49 | 38 | 12 | 8 | 6 |
| | $10^{-1}$ | 0.01 | × | × | × | × | × | 735 | 80 | 38 | 16 |
| | $\infty$ | 0.01 | × | × | × | × | × | × | × | 219 | 35 |
| RANS Re = $10^6$ | $10^{-3}$ | 0.2 | 76 | 70 | 56 | 33 | 30 | 28 | 8 | 9 | 7 |
| | $10^{-1}$ | 0.2 | × | × | × | × | 963 | × | 35 | 30 | 25 |
| | $\infty$ | 0.2 | × | × | × | × | × | × | 70 | 40 | 18 |
| | $10^{-3}$ | 0.01 | 411 | 311 | 231 | 174 | 131 | 110 | 14 | 11 | 9 |
| | $10^{-1}$ | 0.01 | × | × | × | × | × | × | 46 | 28 | 16 |
| | $\infty$ | 0.01 | × | × | × | × | × | × | 132 | 45 | 28 |

**6.3.1. Convergence results.** The convergence results for the Navier–Stokes model for a range of Reynolds numbers, Mach numbers, timesteps $\Delta t$, and preconditioners are presented in Table 6.3. Apart from the inviscid problem with $M = 0.2$ and $\Delta t = 10^{-3}$, all problems have CFL numbers higher than 100,000, which makes an explicit solution procedure impractical.

In general, we see similar trends as for the convection-diffusion system, but we make the following observations:

- The block Jacobi preconditioner performs very poorly in general, showing decent performance only for small timesteps and large Mach numbers. This does not improve significantly when used with a coarse grid correction, with the exception of the inviscid problem for $M = 0.2$. This result is consistent with the the findings reported in [31].
- The block GS preconditioner behaves similarly to the block Jacobi preconditioner, with only about a factor of 2 faster convergence. This shows that the optimal convergence for the pure convection example before was an exception and that GS is highly unreliable for more complex problems.
- The ILU0 preconditioner shows very good performance for almost all of the test cases, except for large timesteps and small Mach numbers. In these cases, the coarse grid correction results are dramatic improvements.

Table 6.4

*The convergence of the BILU0-p0 preconditioner for the Navier–Stokes problem with varying Reynolds numbers, with $M = 0.2$ and $\Delta t = 1.0$. In all problem regimes our MDF results in very significant improvements.*

| Problem | Element ordering | | |
|---|---|---|---|
| | Random | RCM | MDF |
| Inviscid | 51 | 27 | 12 |
| Laminar, Re = 1,000 | 200 | 135 | 12 |
| Laminar, Re = 20,000 | 197 | 139 | 27 |
| RANS, Re = $10^6$ | 98 | 99 | 18 |

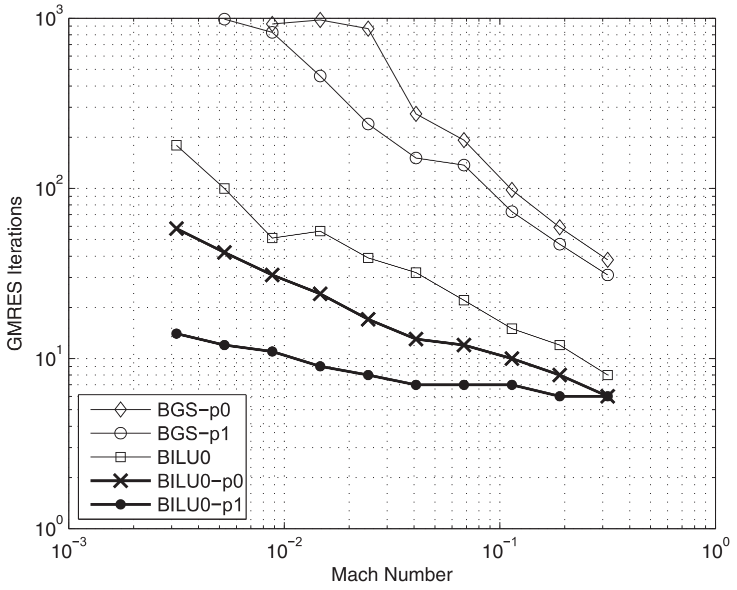


Fig. 6.7. *Convergence of GMRES(20) for the Navier–Stokes problems, as a function of the Mach number. The convergence deteriorates with decreasing Mach number for all methods except for BILU-p1, which is very insensitive to Mach number changes.*

**6.3.2. ILU0 ordering dependencies.** It turns out that our MDF ordering algorithm is also essential for Navier–Stokes problems, as shown in Table 6.4. Here, we set $M = 0.2$ and $\Delta t = 1.0$ and solve the four problems with the BILU0-p0 solver for three element orderings: random, RCM, and the MDF algorithm. We see differences similar to those for the pure convection problems, with up to a magnitude fewer iterations with MDF than with a random ordering. The Euler problem is the least sensitive to ordering, where the simpler RCM algorithm gives convergence about twice as slow as MDF gives.

**6.3.3. Mach number dependence.** To investigate how the convergence depends on the Mach number in more detail, we solve the inviscid problem with $\Delta t = 0.1$ for a wide range of Mach numbers. The plot in Figure 6.7 shows again that for all the methods, the performance worsens as the Mach number decreases, except for the BILU0-p1 preconditioner and to some extent the BILU0-p0. With BILU0-p1, the number of GMRES iterations is almost independent of the Mach number, which is remarkable. This is clearly due to the fact that block ILU0 is an efficient smoother; that is, it reduces the high frequency errors both within and between the elements. Combined with a coarse scale correction, which reduces the low frequency errors, this results in a highly efficient approximate solver. However, the ILU0 solver by itself or the coarse scale correction with GS do not perform as well since they reduce the error in only some of the modes. This provides some intuition as to why the BILU0-p1 preconditioner performs well, but a more quantitative understanding of its properties will require further analysis.

**7. Conclusions.** We have presented a study of preconditioners for DG discretizations of the compressible Navier–Stokes equations. We have found that many of the commonly used preconditioners, such as block Jacobi, block GS, and multilevel schemes, do not perform well in most flow regimes. Therefore, we propose a preconditioner based on a coarse scale correction with an incomplete LU factorization with zero fill-in as a smoother. The element ordering is critical for incomplete factorizations, and we propose a greedy-type heuristic algorithm that can improve the convergence by orders of magnitude. Our preconditioner performs consistently very well for a large range of Reynolds numbers, Mach numbers, and timesteps, for isotropic and highly anisotropic meshes, and for other equations such as RANS with turbulence modeling. A remarkable fact about the proposed preconditioner is that the convergence is largely independent of the Mach number for values almost as low as $M = 10^{-3}$. For this situation, the other preconditioners considered are usually magnitudes slower.

## REFERENCES

[1] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *PETSc home page*, http://www.mcs.anl.gov/petsc.

[2] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1993.

[3] F. Bassi and S. Rebay, *A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations*, J. Comput. Phys., 131 (1997), pp. 267–279.

[4] M. Benzi, D. B. Szyld, and A. van Duin, *Orderings for incomplete factorization preconditioning of nonsymmetric problems*, SIAM J. Sci. Comput., 20 (1999), pp. 1652–1670.

[5] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[6] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[7] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM, Philadelphia, 2000.

[8] B. Cockburn and C.-W. Shu, *The local discontinuous Galerkin method for time-dependent convection-diffusion systems*, SIAM J. Numer. Anal., 35 (1998), pp. 2440–2463.

[9] B. Cockburn and C.-W. Shu, *Runge–Kutta discontinuous Galerkin methods for convection-dominated problems*, J. Sci. Comput., 16 (2001), pp. 173–261.

[10] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 944–961.

[11] J., Douglas, Jr., and T. Dupont, *Interior penalty procedures for elliptic and parabolic Galerkin methods*, in Computing Methods in Applied Sciences, Lecture Notes in Phys. 58, Springer-Verlag, Berlin, 1976, pp. 207–216.

[12] H. C. Elman, V. E. Howle, J. N. Shadid, and R. S. Tuminaro, *A parallel block multi-level preconditioner for the 3D incompressible Navier–Stokes equations*, J. Comput. Phys., 187 (2003), pp. 504–523.

[13] C. Farhat, N. Maman, and G. W. Brown, *Mesh partitioning for implicit computations via iterative domain decomposition: Impact and optimization of the subdomain aspect ratio*, Internat. J. Numer. Methods Engrg., 38 (1995), pp. 989–1000.

[14] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal, *p-multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations*, J. Comput. Phys., 207 (2005), pp. 92–113.

[15] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1981.

[16] A. George and J. W. H. Liu, *The evolution of the minimum degree ordering algorithm*, SIAM Rev., 31 (1989), pp. 1–19.

[17] J. Gopalakrishnan and G. Kanschat, *A multilevel discontinuous Galerkin method*, Numer. Math., 95 (2003), pp. 527–550.

[18] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, *High-performance parallel implicit CFD*, Parallel Comput., 27 (2001), pp. 337–362.

[19] W. Hackbusch, *Multigrid Methods and Applications*, Springer Series in Computational Mathematics 4, Springer-Verlag, Berlin, 1985.

[20] P. Hénon and Y. Saad, *A parallel multistage ILU factorization based on a hierarchical graph decomposition*, SIAM J. Sci. Comput., 28 (2006), pp. 2266–2293.

[21] M. Heroux and A. Williams, *AztecOO*, http://trilinos.sandia. gov/packages/aztecoo.

[22] J. S. Hesthaven and T. Warburton, *Nodal high-order methods on unstructured grids.* I. *Time-domain solution of Maxwell's equations*, J. Comput. Phys., 181 (2002), pp. 186–221.

[23] A. Jameson and S. Yoon, *Lower-upper implicit schemes with multiple grids for the Euler equations*, AIAA J., 25 (1987), pp. 929–935.

[24] G. Kanschat, *Robust smoothers for high order discontinuous Galerkin discretizations of advection-diffusion problems*, J. Comput. Appl. Math., to appear.

[25] D. A. Knoll and D. E. Keyes, *Jacobian-free Newton–Krylov methods: A survey of approaches and applications*, J. Comput. Phys., 193 (2004), pp. 357–397.

[26] D. A. Knoll and W. J. Rider, *A multigrid preconditioned Newton–Krylov method*, SIAM J. Sci. Comput., 21 (1999), pp. 691–710.

[27] T. H. Koornwinder, *Askey–Wilson polynomials for root systems of type BC*, in Hypergeometric Functions on Domains of Positivity, Jack Polynomials, and Applications, Contemp. Math 138., AMS, Providence, RI, 1992, pp. 189–204.

[28] I. Lee, P. Raghavan, and E. G. Ng, *Effective preconditioning through ordering interleaved with incomplete factorization*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1069–1088.

[29] D. J. Mavriplis, *An assessment of linear versus nonlinear multigrid methods for unstructured mesh solvers*, J. Comput. Phys., 175 (2002), pp. 302–325.

[30] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[31] C. R. Nastase and D. J. Mavriplis, *High-order discontinuous Galerkin methods using an hp-multigrid approach*, J. Comput. Phys., 213 (2006), pp. 330–357.

[32] N. C. Nguyen, P.-O. Persson, and J. Peraire, *RANS solutions using high order discontinuous Galerkin methods*, in Proceedings of the 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2007, AIAA-2007-914.

[33] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[34] J. Peraire and P.-O. Persson, *The compact discontinuous Galerkin (CDG) method for elliptic problems*, SIAM J. Sci. Comput., 30 (2008), pp. 1806–1824.

[35] P.-O. Persson and J. Peraire, *An efficient low memory implicit DG algorithm for time dependent problems*, in Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2006, AIAA-2006-0113.

[36] P. L. Roe, *Approximate Riemann solvers, parameter vectors, and difference schemes*, J. Comput. Phys., 43 (1981), pp. 357–372.

[37] E. M. Rønquist and A. T. Patera, *Spectral element multigrid.* I. *Formulation and numerical results*, J. Sci. Comput., 2 (1987), pp. 389–406.

[38] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.

[39] J. N. Shadid, R. S. Tuminaro, K. D. Devine, G. L. Hennigan, and P. T. Lin, *Performance of fully coupled domain decomposition preconditioners for finite element transport/reaction simulations*, J. Comput. Phys., 205 (2005), pp. 24–47.

[40] L. F. Shampine and C. W. Gear, *A user's view of solving stiff ordinary differential equations*, SIAM Rev., 21 (1979), pp. 1–17.

[41] P. Wesseling, *A robust and efficient multigrid method*, in Multigrid Methods, Lecture Notes in Math. 960, Springer-Verlag, Berlin, 1982, pp. 614–630.

[42] G. Wittum, *On the robustness of ILU-smoothing*, in Robust Multi-Grid Methods, Notes Numer. Fluid Mech. 23, Vieweg, Braunschweig, 1989, pp. 217–239.