INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2006; **51**:1157–1176
Published online 20 December 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.1135

Hierarchic multigrid iteration strategy for the discontinuous Galerkin solution of the steady Euler equations

Koen Hillewaert[1,*,†], Nicolas Chevaugeon[2], Philippe Geuzaine[1] and Jean-François Remacle[2]

[1]*CENAERO, CFD-Multiphysics Group, Bâtiment Mermoz 1, Av. J. Mermoz 30, B-6041 Gosselies, Belgium*
[2]*Département du Génie Civil et Environnemental, Université Catholique de Louvain, Bâtiment Vinci, Place du Levant, B-1348 Louvain-La-Neuve, Belgium*

SUMMARY

We study the efficient use of the discontinuous Galerkin finite element method for the computation of steady solutions of the Euler equations. In particular, we look into a few methods to enhance computational efficiency. In this context we discuss the applicability of two algorithmical simplifications that decrease the computation time associated to quadrature. A simplified version of the quadrature free implementation applicable to general equations of state, and a simplified curved boundary treatment are investigated. We as well investigate two efficient iteration techniques, namely the classical Newton–Krylov method used in computational fluid dynamics codes, and a variant of the multigrid method which uses interpolation orders rather than coarser tesselations to define the auxiliary coarser levels. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: discontinuous Galerkin; Euler equations; quadrature free; multigrid; Newton–Krylov method

1. INTRODUCTION

The discontinuous Galerkin finite element method (DGFEM) has become a very popular method for the computation of unsteady flows, especially when accurate solutions are needed.

This is due to the combination of an arbitrary order of accuracy with data and algorithmic locality. Since the interpolation functions are defined independently in each element the mass matrix has a block-diagonal structure and may be inverted directly. The method can be easily

*Correspondence to: K. Hillewaert, CENAERO, CFD-Multiphysics Group, Bâtiment Mermoz 1, Av. J. Mermoz 30, B-6041 Gosselies, Belgium.
†E-mail: koen.hillewaert@cenaero.be

Contract/grant sponsor: CENAERO is funded by the Walloon Region and the structural funds ERDF and ESF; contract/grant number: EP1A122030000102

Received 2 June 2005
Revised 13 October 2005
Accepted 20 October 2005

parallelized because DGFEM stencils do not grow in size with increasing order. The interpolation functions are not restrained by continuity requirements and can hence be chosen freely, allowing easy implementation of, e.g. spectral interpolation bases. Another consequence is the inherent capability of the method to handle h- and p-adaptation.

The method has not experienced a similar interest for the computation of steady flows. On one hand, the accuracy requirements are not as stringent for these computations. One does not care about temporal dispersion and dissipation, so one can use mesh refinement without too severe consequences on computational efficiency. On the other hand, DGFEM still has some evident drawbacks in terms of computational efficiency. First of all an efficient iteration method is still lacking. Furthermore, the method seems to be very expensive in terms of memory (more unknowns for the same formal order) and work load due to quadrature. Both are exacerbated by the apparent need for isoparametric elements near curved boundaries.

Fidkowski *et al.* have recently shown [1] that in terms of CPU time for a given precision it is better to increase DGFEM solution order rather than mesh resolution, at least for smooth solutions. Hence it is likely that the high-order DGFEM method will grow to be more efficient than a state-of-the-art finite volume solver for steady computations as well, especially if it is combined with h- and p-adaptation. Another advantage is that accurate results can be obtained with—by today's standards—very coarse meshes, thus relaxing the requirements for the mesh generators (which often are not parallelized and are hence limited in mesh size) and diminishing the overhead related to storage of mesh topology.

In this paper, we investigate methods for increasing computational efficiency in terms of computing time and memory requirements, both by algorithmical simplifications and efficient iteration strategies.

The paper is organized as follows. First DGFEM is reviewed briefly. After that we discuss the application of two algorithmical simplifications of DGFEM which decrease both computing time and memory requirements for the evaluation of the discretization residual. In the following section we discuss and compare several efficient iteration strategies for stationary DGFEM.

2. THE DISCONTINUOUS GALERKIN METHOD

We focus on the application of DGFEM to approximate the steady solutions u to the Euler equations of gas dynamics in a domain $\mathscr{D}$. We note the Euler equations concisely as

$$\frac{\partial u}{\partial t} + \nabla \cdot \vec{f}(u) = 0 \tag{1}$$

DGFEM approximates the solution u in the broken or discontinuous interpolation space $\mathscr{U}$ that is spanned by shape functions ϕ_i. Each of these functions ϕ_i is defined on one element T only of a tesselation $\mathscr{T}$ of domain $\mathscr{D}$. The approximate solution $\tilde{u}$ is as such discontinuous across element boundaries.

$$\tilde{u} = \sum_i \mathbf{u}_i \phi_i, \quad \phi_i \in \mathscr{U} \tag{2}$$

The expansion coefficients $\mathbf{u}_i$ are found by requiring that the approximate solution $\tilde{u}$ satisfies the following weak formulation of (1):

$$\left(\frac{\partial \tilde{u}}{\partial t} + \nabla \cdot \vec{f}(\tilde{u}), v \right) = 0, \quad \forall v \in \mathscr{U} \tag{3}$$

where the inner product in the broken space $\mathscr{U}$ has been defined as

$$(p,q) = \int_{\mathscr{D}} p\,q\,\mathrm{d}V \quad \forall p,q \in \mathscr{U} \tag{4}$$

Considering that any $v \in \mathscr{U}$ is a linear combination of shape functions ϕ_i, and any shape function ϕ_i is only supported on one element T we can rewrite (3) as

$$\begin{aligned}
&\int_{\mathscr{D}} \phi_j \frac{\partial \tilde{u}}{\partial t}\,\mathrm{d}V + \int_{\mathscr{D}} \phi_j \nabla \cdot \vec{f}(\tilde{u})\,\mathrm{d}V = 0 \quad \forall \phi_j \in \mathscr{U} \\
&= \int_T \phi_j \frac{\partial \tilde{u}}{\partial t}\,\mathrm{d}V + \oint_{\partial T} \phi_j \vec{f}(\tilde{u}) \cdot \vec{n}\,\mathrm{d}S - \int_T \nabla \phi_j \cdot \vec{f}(\tilde{u})\,\mathrm{d}V \\
&\approx \int_T \phi_j \frac{\partial \tilde{u}}{\partial t}\,\mathrm{d}V + \oint_{\partial T} \phi_j \mathscr{H}(\tilde{u}^+, \tilde{u}^-, \vec{n})\,\mathrm{d}S - \int_T \nabla \phi_j \cdot \vec{f}(\tilde{u})\,\mathrm{d}V
\end{aligned} \tag{5}$$

Since the solution is discontinuous across element boundaries, the boundary flux $\vec{f}(\tilde{u}) \cdot \vec{n}$ corresponds to the solution of a local Riemann problem. This flux is thus replaced by a numerical flux function $\mathscr{H}$, depending on the solutions $\tilde{u}^+$ and $\tilde{u}^-$ approaching the boundary from the inside and outside, respectively, and the boundary normal $\vec{n}$. If this flux function is positive, it provides the correct amount of stabilization. The effect of the stabilization is the damping of modes that are unresolved by the chosen discretization. For the Euler equations we typically use the flux difference splitting scheme of Roe [2].

3. ALGORITHMIC IMPROVEMENTS

The DGFEM is a relatively expensive method since more unknowns are required for the same formal order than for instance high order continuous finite elements. Moreover, exact quadrature involved in the correct integration of the weak form is even more penalizing. The number of term of each block of the system grows as p^2 in two dimensions (p^3 in 3). Each term is computed as the integral of a polynomial of order $2p+1$ and the number of quadrature points for evaluating that term then grows like $(2p+1)^2$ ($(2p+1)^3$ for three dimensions). Furthermore, for each quadrature point on element-interfaces, an expensive flux function needs to be computed. The apparent necessity for (near)isoparametric elements adjacent to solid domain boundaries not only complicates the method but also increases the number of quadrature points. Hence, the simplification and acceleration of the quadrature method is an important step towards an efficient method.

3.1. Boundary representation

Bassi *et al.* [3] have shown that an accurate boundary representation is necessary to maintain the formal order of the scheme and to avoid spurious production of entropy at the intersection of boundary faces. They stated that an equal order interpolation of solution and coordinates for elements adjacent to the boundaries is unavoidable. This requires the use of a higher number of quadrature points to account for the variation of the metric.

In a novel approach, suggested by Baumann *et al.* [4], the elements adjacent to the domain boundaries remain simplices but an accurate representation of the boundary normal is used in

the quadrature points. Krivodonova *et al.* [5] subsequently implemented different variants of this approach and showed numerically that the formal order of the method is well conserved. Using only simplices instead of boundary fitted elements represents a huge simplification of the code, and tremendous savings in memory and computing time.

In our implementation the normal to be applied at the Gauss points is found by projecting the Gauss point on the true surface, which is described by an analytical expression. The projection uses an iterative procedure such that it's direction is aligned to the computed normal.

3.2. Quadrature free implementation

In Reference [6] Lockard *et al.* further elaborate the *quadrature free* implementation of DGFEM as put forward by Atkins *et al.* [7]. This method avoids explicit quadrature and results in a significant reduction of computational time. Since this method uses only expansion weights, and does not explicitly reconstruct values at the quadrature points, the number of flux evaluations is drastically reduced. Furthermore, all other operations may be cast in matrix–vector and matrix–matrix products for which highly optimized libraries exist.

The method starts from the expansion of the flux in terms of the base functions:

$$\vec{f}(\tilde{u}) = \sum_i \vec{\mathbf{f}}_i \phi_i \tag{6}$$

If we assume the elements to be simplices, the metric of the element is constant. With these assumptions the volume term can then be rewritten as

$$\begin{aligned} \int_T \nabla\phi_j \vec{f}(\tilde{u})\,\mathrm{d}V &\approx \vec{\mathbf{f}}_i \left(\int_T \nabla\phi_j \phi_i \,\mathrm{d}V \right) \\ &= \vec{\mathbf{f}}_i \cdot \mathbf{J}_T^{-1} \cdot \left(\int_{\widehat{T}} \widehat{\nabla}\phi_j \phi_i \,\mathrm{d}\xi\,\mathrm{d}\eta \right) J_T \\ &= \vec{\mathbf{f}}_i \cdot \mathbf{J}_T^{-1} \cdot \vec{C}_{ij} J_T \end{aligned} \tag{7}$$

where $\mathbf{J}_T$ is the Jacobian of the mapping of the element T onto the reference element and J_T is its determinant; $\widehat{\nabla}$ defines the gradient in parameter space:

$$\mathbf{J}_T = \frac{\partial(x, y)}{\partial(\xi, \eta)} \tag{8}$$

$$\widehat{\nabla} a = \frac{\partial a}{\partial \xi_k} \vec{e}_{\xi_k} \tag{9}$$

The vector of matrices $\vec{C}$ is independent of the element, and may be precalculated for any order. For each element T, we need to precalculate the metric terms $\mathbf{J}_T^{-1}$ and J_T. Once the expansion coefficients $\vec{\mathbf{f}}_i$ have been determined, the evaluation of the volume term is reduced to matrix–matrix and matrix–vector products of relatively small dimensions. A similar development may be done for the surface term of the boundary flux. The weights $\vec{\mathbf{f}}_i$ are found trivially if the flux vector is a linear function of the approximate solution $\tilde{u}$. Atkins proposes an elegant method to project the fluxes of the Euler equations for a perfect gas. Unfortunately, a similar method is difficult to find for generic flux functions, generic equations of state, or

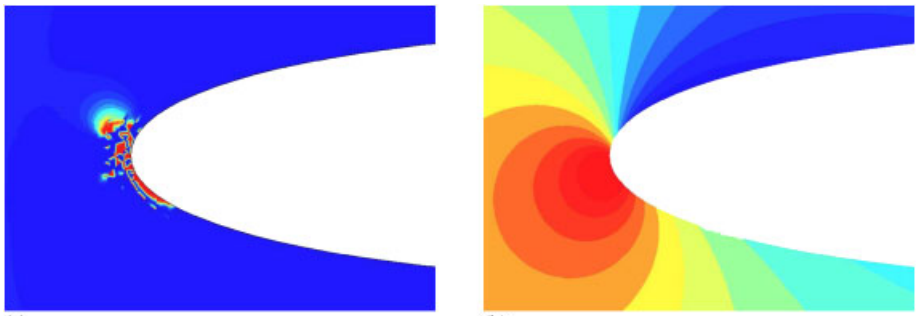

Figure 1. Singular point instability: (a) quadrature free; and (b) full quadrature.

even for the expansion of the boundary flux $\mathscr{H}$ of the perfect gas Euler equations. Given the computational cost of the evaluation of the boundary function in case of the Euler equations, the quadrature free method loses much of its interest if it does not apply to the boundary terms as well.

If we use a nodal expansion basis we may however assume that a function which passes through the nodal values of the flux vector is a reasonable approximation of the flux expansion in the volume as well as on the boundary. The expansion is then trivial, since we compute the nodal weights $\vec{\mathbf{f}}_i$ pointwise from $\mathbf{u}_i$:

$$\vec{\mathbf{f}}_i = \vec{f}(\mathbf{u}_i) \tag{10}$$

Apart from the apparent deviation of the formal order of the scheme, a major problem of this approach is that the interpolation weight vectors $\mathbf{u}_i$ are only weakly coupled numerically. This is because the projection of the flux (10) uses flux values that are computed using only one expansion coefficient vector $\mathbf{u}_i$. When the flux becomes singular, which is the case at stagnation and sonic points, the weights $\mathbf{u}_i$ may vary randomly without modifying the computed flux. Using normal quadrature eliminates this problem since any degree of freedom (DOF) is used for flux evaluations at different Gauss integration points. The instability is illustrated in Figure 1, which shows a zoom of the leading edge region of the NACA0012 airfoil. The free-stream conditions are $M = 0.3$ and angle of attack of 2° and the solution has been approximated with second-order polynomials; a Newton–Krylov implicit iteration scheme has been used for the solution of the discretized equations. The quadrature free solution is obtained just before explosion of the computation.

A very simple fix consists in reverting to classical quadrature in elements where the equations are (nearly) singular, i.e. in which stagnation or sonic points occur. For simple subsonic flows, such as presented in this paper, stagnation points occur exclusively on solid boundaries. A first obvious step is hence to use full quadrature consistently on all solid boundaries. This fix does not penalize overall efficiency, since the number of affected cells is small and we can easily *a priori* group all elements using quadrature free or full quadrature integration, respectively. Furthermore, boundary elements already need a special treatment to take boundary curvature into account into the integration.

For more general flows, and especially for transonic and subsonic flows, an indicator based on the annihilation of one of the characteristic speeds should suffice. This strategy however

introduces a switch-like behaviour, so one needs to look into a robust integration into the (implicit) iteration strategy. Furthermore, some efficiency loss will occur since the elements in which we need to apply full quadrature are not known *a priori*. These aspects have not yet been investigated since the goal here was to demonstrate the instability and identify its source.

4. ITERATION STRATEGIES

One of the main obstacles towards using DGFEM for stationary computations is the lack of an efficient iteration strategy. DGFEM has up to now been used primarily for the explicit and fully resolved computation of unsteady flows. To compute steady flows those explicit time-integrators are prohibitively slow, even when using locally defined optimal timesteps.

4.1. Newton–Krylov

The Newton–Krylov method is fairly often used in finite volume methods [8–10] and has only recently been applied to DGFEM [11, 12]. In this method the discretized non-linear set of equations is solved using Newton iterations. The linear system arising from Newton linearizations is iteratively solved with a Krylov subspace iteration method. In this case we use the classical combination of GMRES with ILU(0) as preconditioner. In this method we add a pseudo-time derivative to the original residual to improve the conditioning of the linear system:

$$\left(\frac{\tilde{u}^n - \tilde{u}^{n-1}}{\Delta\tau^n} + \nabla\cdot\vec{f}(\tilde{u}^n), v\right) = 0 \tag{11}$$

The pseudo-timestep $\Delta\tau^n$ is chosen locally to conform to a given CFL number, and thus provide the 'locally optimal' diagonal dominance. Since this amounts to solving a pseudo-unsteady problem the non-linear convergence is governed by the pseudo-timestep, and will be obviously faster as $\Delta\tau^n$ becomes larger. Therefore, the CFL is updated following each Newton update, inversely proportional to a power α of the current reduction of the L2-norm of the residual ε:

$$\mathrm{CFL}_n = \max\left(\mathrm{CFL}_0\cdot\left(\frac{\|\varepsilon_0\|_2}{\|\varepsilon_n\|_2}\right)^{\alpha}, \mathrm{CFL}_{\max}\right) \tag{12}$$

A major drawback of the Newton–Krylov-ILU method for high-order DGFEM is the large memory footprint. The elementary blocks composing the preconditioning matrix increase quadratically in size with the number of unknowns per element/face. If p is the polynomial order, then the number of unknowns per element grows like p^2 in 2 and p^3 in three dimensions. Then, the number of elements in each block of the ILU grows like p^4 resp. p^6. Hence a matrix preconditioner becomes quite impractical even for relatively low orders.

A second problem concerns the application of the slope limiter. Classical techniques of flux limiting are not applicable for high order DGFEM because of the presence of volume terms in the formulation. Hence the slope limiter is not integrated in the computation of the residual, but effectively acts as a post-processing filter [13]. Such a filter is easily integrated in an explicit, but not into an implicit iteration strategy, such as the Newton–Krylov method.

Finally, it is necessary to tune the evolution of the CFL number in order to maintain sufficient diagonal dominance throughout the computation. This becomes particularly difficult for higher orders.

4.2. Multigrid

With most iterative methods more iterations are needed to converge as the number of degrees of freedom increases. This is due to either a poorer conditioning of the global system for implicit iterative solution methods, or smaller stable timesteps in the case of explicit time integration.

The multigrid method overcomes this problem by using a sequence of ever coarser representations of the discretized problem, in the hope of keeping the same convergence rate per iteration as for the coarsest representation. If the work needed to solve the problem is directly proportional to the number of degrees of freedom, one attains the *textbook multigrid efficiency* (TME). TME requires constant residual reduction per multigrid cycle, independent of the finest representation of the solution. Multigrid methods succeed in doing this because

- High frequency errors tend—as a general trend—to be well noticed by a discretization method. Hence iterative solution methods tend to reduce (dissipate) those errors more efficiently. Multigrid uses successive coarser representations of the problem to convert low frequency errors to high frequency errors on those coarser levels, thus leading to a similarly fast elimination of the low frequency errors.
- The relaxation parameters (e.g. a pseudo-timestep) have higher stable/convergent values with respect to the physical dimensions on the coarser levels, permitting faster evolution of the transient.
- The overhead in terms of work and storage on the coarser levels is negligible in comparison to the finest level.

These features allow multigrid to achieve competitive convergence rates with relatively simple iteration methods. Multigrid is as a consequence a very efficient method, with respect to both memory and computing time, especially when a large number of degrees of freedom is considered.

Classical multigrid methods use different, successively coarser tesselations of the domain. This tends to lead to complicated methods for the transfer of solutions between different representations and *ad hoc* definitions of the defect correction equations. The complexity of the implementation is even more critical in parallel. Classical multigrid has recently been applied to DGFEM discretizations for the groundwater equations by Bastian *et al.* [14]. In the context of hierarchic higher-order methods such as DGFEM, one may as well use lower order interpolation spaces based on the same tesselation to define the coarser representations of the problem. The advantages are the ease of implementation and the possibility to define a firm theoretical basis for the definition of transfer operators and the defect correction equations. A similar technology has been developed independently by Fidkowski *et al.* [1]. The rationale for finding the interlevel transfer operators presented in the current work is however more general and is applicable to both p-multigrid and classical multigrid for any weak formulation using discontinuous interpolation, even when interpolation spaces are not nested.

4.2.1. Full approximation storage. In the basic form of multigrid, the two-level method, we transfer the current fine level iterant to a coarser level. On the coarser level a defect correction is computed, which is then transferred back to the finer representation. The *full approximation storage* (FAS) method is the variant of this scheme for non-linear equations. The description of the basic FAS cycle may be found in the classic textbook [15].

We have redefined the FAS cycle for the case of hierarchic multiorder or p-multigrid iterations as follows. Suppose we want to find the approximated solution $\tilde{u}^p$ to (1) in the space $\mathscr{U}^p$ of order p, based on the tesselation $\mathscr{T}$. First we define the functional residual operator $\mathscr{L}()$ as

$$\mathscr{L}(\tilde{u}) = \nabla \cdot \vec{f}(u) \tag{13}$$

Consider then another interpolation space $\mathscr{U}^q$ which is based on the same tesselation $\mathscr{T}$, and where $q < p$. The two-level FAS cycle then proceeds as follows:

1. *Pre-smoothing*: perform a number of iterations on level p, leading to solution $\tilde{u}^{p*}$.
2. *Restriction*: approximate $\tilde{u}^{p*}$ by $\tilde{u}^{q*}$ in $\mathscr{U}^q$.
3. *Defect correction*: solve exactly in $\mathscr{U}^q$:

$$(\mathscr{L}(\tilde{u}^q) - \mathscr{L}(\tilde{u}^{q*}) + \mathscr{L}(\tilde{u}^{p*}), v^q) = 0 \quad \forall v^q \in \mathscr{U}^q \tag{14}$$

4. *Prolongation*: represent the correction $\tilde{u}^q - \tilde{u}^{q*}$ in $\mathscr{U}^p$ and add to $\tilde{u}^{p*}$:

$$\tilde{u}^p = \tilde{u}^{p*} + P_q^p(\tilde{u}^q - \tilde{u}^{q*}) \tag{15}$$

5. *Post-smoothing*: perform additional iterations on level p to smooth the corrected solution.

We can interpret the function of this implementation of the defect correction equation in the following manner: we eliminate the part of the residual which is representable in the lower order space. Suppose $\mathscr{U}^q \subset \mathscr{U}^p$, then we define

$$\mathscr{U}^{p-q} := \mathscr{U}^p \backslash \mathscr{U}^q \tag{16}$$

We may then decompose $\tilde{u}^p$

$$\tilde{u}^p = \tilde{u}^q + \tilde{u}^{p-q}, \quad \tilde{u}^q \in \mathscr{U}^q, \ \tilde{u}^{p-q} \in \mathscr{U}^{p-q} \tag{17}$$

If the Galerkin weighted residuals are linear in the expansion coefficients, we may rewrite the defect correction weighted residual (14) as

$$(\mathscr{L}(u^{(p-q)*} + (\tilde{u}^q - \tilde{u}^{q*})), v^q) = 0 \quad \forall v^q \in \mathscr{U}^q \tag{18}$$

We see that the defect correction equation defines the lower order correction to the solution that removes the corresponding order variations of the residual.

Since both interpolation spaces use the same elements, the definition of the defect residual and the prolongation/restriction are trivial. The prolongation and restriction of the solutions both use Galerkin or L_2-projection. This proceeds as follows: let us consider a solution $\tilde{u}^a \in \mathscr{U}^a$ and its Galerkin projection $\mathscr{I}_a^b \tilde{u}^a = \tilde{u}^b \in \mathscr{U}^b$

$$\begin{aligned} \tilde{u}^a &= \sum_i \mathbf{u}_i^a \phi_i^a, \quad \phi_i^a \in \mathscr{U}^a \\ \mathscr{I}_a^b \tilde{u}^a = \tilde{u}^b &= \sum_j \mathbf{u}_j^b \phi_j^b, \quad \phi_j^b \in \mathscr{U}^a \end{aligned} \tag{19}$$

Orthogonalizing the difference $\tilde{u}^a - \tilde{u}^b$ to the space $\mathscr{U}^b$ defines the following set of equations for the expansion coefficients $\mathbf{u}_i^b$:

$$\sum_i (\phi_k^b, \phi_j^b)\mathbf{u}_j^b = \sum_j (\phi_k^b, \phi_i^a)\mathbf{u}_i^a \quad \forall \phi_k^b \in \mathscr{U}^b \tag{20}$$

The solution transfer operator $\mathscr{I}_a^b$ has a discrete or matrix equivalent I_a^b defining the transfer between the expansion vectors $\mathbf{u}^a = [\mathbf{u}_1^a \dots \mathbf{u}_{n_a}^a]^{\mathrm{T}}$ and $\mathbf{u}^b = [\mathbf{u}_1^b \dots \mathbf{u}_{n_b}^b]^{\mathrm{T}}$

$$\begin{aligned} \mathbf{u}^b &= I_a^b \cdot \mathbf{u}^a \\ I_a^b &= (M^b)^{-1} \cdot M^{ba} \end{aligned} \tag{21}$$

where

$$\begin{aligned} M_{ij}^b &= (\phi_i^b, \phi_j^b) \\ M_{ij}^{ba} &= (\phi_i^b, \phi_j^a) \end{aligned} \tag{22}$$

The 'restriction' of the residual vector follows directly from the weighted defect correction equation. Conventionally one goes the other way around: first a restriction operator is defined for the residual vector, and then—in the best of cases—the coarse grid operator is found by applying the discrete operators to the fine grid operator (Galerkin coarse grid approximation) or—more frequently—one uses the same discretization technique on the coarse representation (discrete coarse grid approximation). The definition of the forcing term requires the computation of the 'restricted' residual

$$(\mathscr{L}(\tilde{u}^p), v^q) \tag{23}$$

To compute this term explicitly we would need to redefine routines for weighting all terms of the residual defined in $\mathscr{U}^p$ with test functions in $\mathscr{U}^q$. To avoid this complication, we first expand $\mathscr{L}(\tilde{u}^p)$ in $\mathscr{U}^p$:

$$\begin{aligned} \mathscr{L}(\tilde{u}^p) &\approx \sum_i r_i^p \phi_i^p \\ \sum_i r_i^p (\phi_i^p, \phi_j^p) &\approx (\mathscr{L}(\tilde{u}^p), \phi_j^p) \end{aligned} \tag{24}$$

This projection requires the mass matrix and the Galerkin weighted residual defined on space $\mathscr{U}^p$, both of which are already available. The 'restricted residual' is then computed as

$$(\mathscr{L}(\tilde{u}^p), v^q) \approx \left(\sum r_i^p \phi_i^p, v^q\right) \tag{25}$$

We find the following matrix operator connecting the vector containing Galerkin weighted residuals for space $\mathscr{U}^p$ to the restricted residuals in space $\mathscr{U}^q$:

$$\tilde{I}_p^q = M^{qp} \cdot (M^p)^{-1} \tag{26}$$

The discrete form of the prolongation and restriction operators as given by Equations (21) and (26) are the same as proposed by Fidkowski *et al.* [1] in the framework of p-multigrid DGFEM for nested interpolation spaces. The rationale of finding the transfer operators indicated here is however more general. In the framework of discontinuous interpolation it

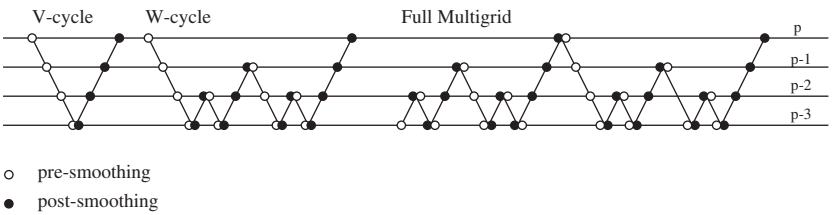

Figure 2. Multigrid strategies.

is equally applicable to non-embedded spaces and even to classical multigrid methods with nested or non-nested grids. Hence this definition may be exploited for hp-adaptive and spectral implementations. Theoretically one may apply it to continuous interpolation spaces as well. However, the transfer operators require the inversion of the mass matrix on both levels, which will probably be prohibitively costly.

4.2.2. Multigrid strategies. The two-grid algorithm requires an accurate solve of the defect correction equation. This may still be untractable, and hence a number of strategies exploiting the two-grid cycle are defined (see Figure 2):

- *Explicit v- and w-cycles*: A first strategy consists in replacing the coarse exact solve by a recursive application of the two-grid cycle. Now we base the restricted residual on the defect correction equation. Depending on the number of two-cycle solves per defect correction we distinguish v-cycles and w-cycles. We define our multigrid cycling strategy by the following parameters:
  - number of coarser levels;
  - number of pre- and post-smoothing steps (or accuracy) for each level;
  - number of subcycles;
  - smoother type for each level.

  Typically we use all available levels, and on each level—unless stated otherwise—we use 10 Runge–Kutta pseudo-time iterations for both the pre- and the post-smoothing step.
- *Partially implicit v- and w-cycles*: Multigrid convergence hinges upon the convergence of the lowest frequency (order) errors which are no longer converted to higher frequencies by coarsening. Returning to the Newton–Krylov implicit solver we observe that, for zeroth order interpolation, the storage implied in the ILU preconditioner is relatively small. The discretization is positive and hence no limiter is required to stabilize the scheme near solution discontinuities. So we have very convincing motivations to use the implicit solver on the coarsest levels.

  Starting the Newton–Krylov iterations from the same CFL each time we get to the coarsest level is not a good idea. The non-linear convergence depends on it, and the required stabilization diminishes as the solution is further converged. Therefore, we update CFL_0 for the Newton–Krylov scheme following each multigrid cycle. We use the same rationale as in (12), but now using the ratios of fine grid residual norms. Stable values for CFL_0, $\mathrm{CFL}_{\max}$ and α may be determined from a zeroth order run.
- *Full multigrid* (*FMG*): Starting from the coarsest representation we successively refine the representation. Using the hitherto available coarser levels one performs a number

of v- or w-cycles. Depending on the strategy a prespecified number of cycles is done, or until a desired level of convergence has been reached. After that the solution is prolongated to the next finer level. This successive refining of the order is also called *nested iteration*. When finally reaching the target order, one continues v- or w-cycling until full convergence. This combination is coined full multigrid.

5. APPLICATIONS

5.1. Grid convergence study: flow around a circular cylinder

To check the grid convergence for both the full quadrature and the quadrature free implementation we consider the computation of the inviscid flow around a circular cylinder at Mach 0.3. We use this testcase since it is easy to generate high quality nested meshes, and because the surface curvature is constant, so there is no discussion about grid clustering. The meshes are the same as those used by Krivodonova *et al.* [5]. The basic grid is fully regular and symmetric, the coarser grids are sequentially nested into the finest mesh. Details of the meshes near the cylinder are displayed in Figure 3. The label of the meshes refers to the number of

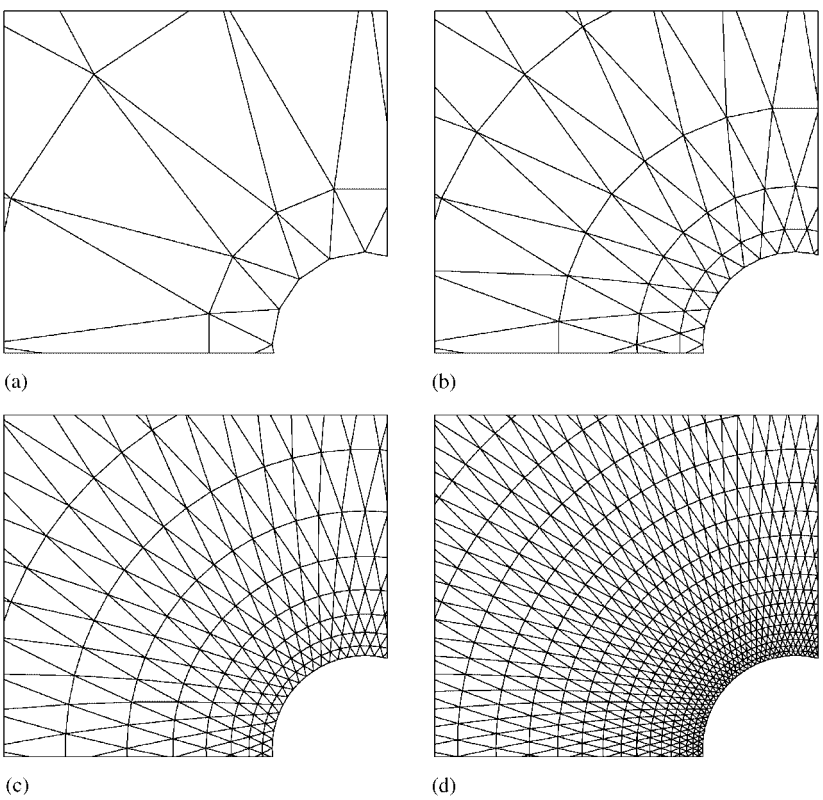

Figure 3. Meshes for the cylinder testcase: (a) 16×4; (b) 32×8; (c) 64×16; and (d) 128×32.

points on the cylinder times the number of parallel mesh layers between the cylinder and the free-stream boundary.

Orders ranging from 0 to 3 have been investigated. The numerical error has been measured by the L2-norm of the following quantity related to entropy (p_∞ and ρ_∞ correspond to the free-stream values) [5]:

$$s' = \left(\frac{p}{p_\infty}\right) \cdot \left(\frac{\rho_\infty}{\rho}\right)^\gamma - 1 \tag{27}$$

The evolution of the error norm is shown in Figure 4. The grid convergence rate r has been computed for each refinement separately and globally, and summarized together with the error norm $\sigma = \mathrm{L}_2(s')$ in Tables I and II. We see that the order of accuracy of the scheme in case of full quadrature corresponds to the results obtained by Krivodonova. The order of

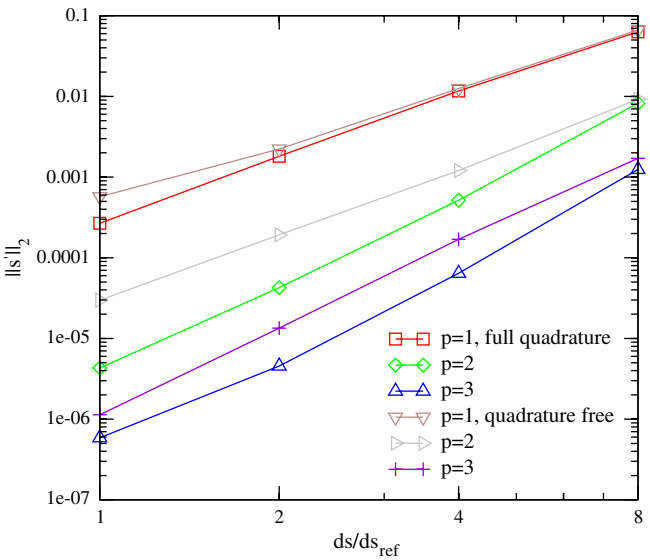

Figure 4. Grid convergence—cylinder testcase.

Table I. Grid convergence—full quadrature—cylinder testcase.

	$p=1$		$p=2$		$p=3$	
	σ	r	σ	r	σ	r
16×4	6.315e − 02	—	8.228e − 03	—	1.244e − 03	—
32×8	1.169e − 02	2.43	5.162e − 04	3.99	6.425e − 05	4.27
64×16	1.812e − 03	2.69	4.272e − 05	3.59	4.552e − 06	3.82
128×32	2.676e − 03	2.76	4.312e − 06	3.31	5.876e − 07	2.95
Global	—	2.63	—	3.63	—	3.68

Table II. Grid convergence—quadrature free—cylinder testcase.

	$p=1$		$p=2$		$p=3$	
	σ	r	σ	r	σ	r
16×4	6.634e − 02	—	9.216e − 03	—	1.707e − 03	—
32×8	1.251e − 02	2.41	1.204e − 03	2.94	1.695e − 04	3.33
64×16	2.226e − 03	2.49	1.924e − 04	2.65	1.343e − 05	3.66
128×32	5.710e − 04	1.96	2.978e − 05	2.69	1.132e − 06	3.57
Global	—	2.29	—	2.76	—	3.52

accuracy and solution accuracy are deteriorated by the quadrature free integration, but not in a very dramatic manner.

Figures 5 and 6 show comparisons of the computed Mach number isolines for both the full quadrature and the patched quadrature free implementation. The Mach number distribution is visualized using 21 isolines between 0 and 0.65.

5.2. Comparison of iterative strategies—subsonic flow around the NACA0012 airfoil

To compare the iteration strategies presented in this paper we compute the flow around the NACA0012 airfoil. The angle of attack is 2° and the Mach number is 0.3. The geometry of the airfoil may be found in Reference [16]. The chord has been rescaled to avoid the truncation of the airfoil at the trailing edge. The free-stream boundary is located at 30 chords from the leading edge. The mesh size on the airfoil varies quadratically, from 0.01 at the leading edge, over 0.08 at midchord to 0.02 at the trailing edge. At the free-stream boundary the grid size is 20. The coarse mesh is obtained by doubling those sizes. The details of the meshes near the airfoil are shown in Figure 7. The coarse mesh contains in total 157 nodes, the fine mesh 622 nodes. Again the simplified treatment for boundary curvature has been used.

Computed Mach number isolines for interpolation orders 1 and 4 on the coarse and on the fine mesh are compared in Figure 8. In Figure 9 we compare performance of three full multigrid strategies in terms of CPU time. All strategies use fully explicit w-cycles and are compared to the standard w-cycle. Residuals are always computed using the solution projected onto the highest order representation. The first strategy, labelled ‘w-fmg1’ only uses one cycle per level during the nested iteration. The second one (‘w-fmg2’) converges four orders of magnitude for each refinement. The last one (‘w-fmg3’) converges each level to machine accuracy. We see that w-fmg1 and w-fmg2 have more or less the same performance as the w-cycle. W-fmg3 performs much worse. This is due to the fact that from a given resolution onward the low order solution generates the same higher-order error. All the time spent in improving the low order solution with respect to this critical resolution is then lost. At least for this case, we have nothing to gain from FMG. The only advantage we may reasonably expect from FMG is an increased stability. Full resolution for each successive refinement is however a waste of CPU time.

In Figures 10 and 11 we compare the convergence rates of the different iteration strategies in terms of number of multigrid cycles (if applicable) and CPU time for orders 2, 3 and 4

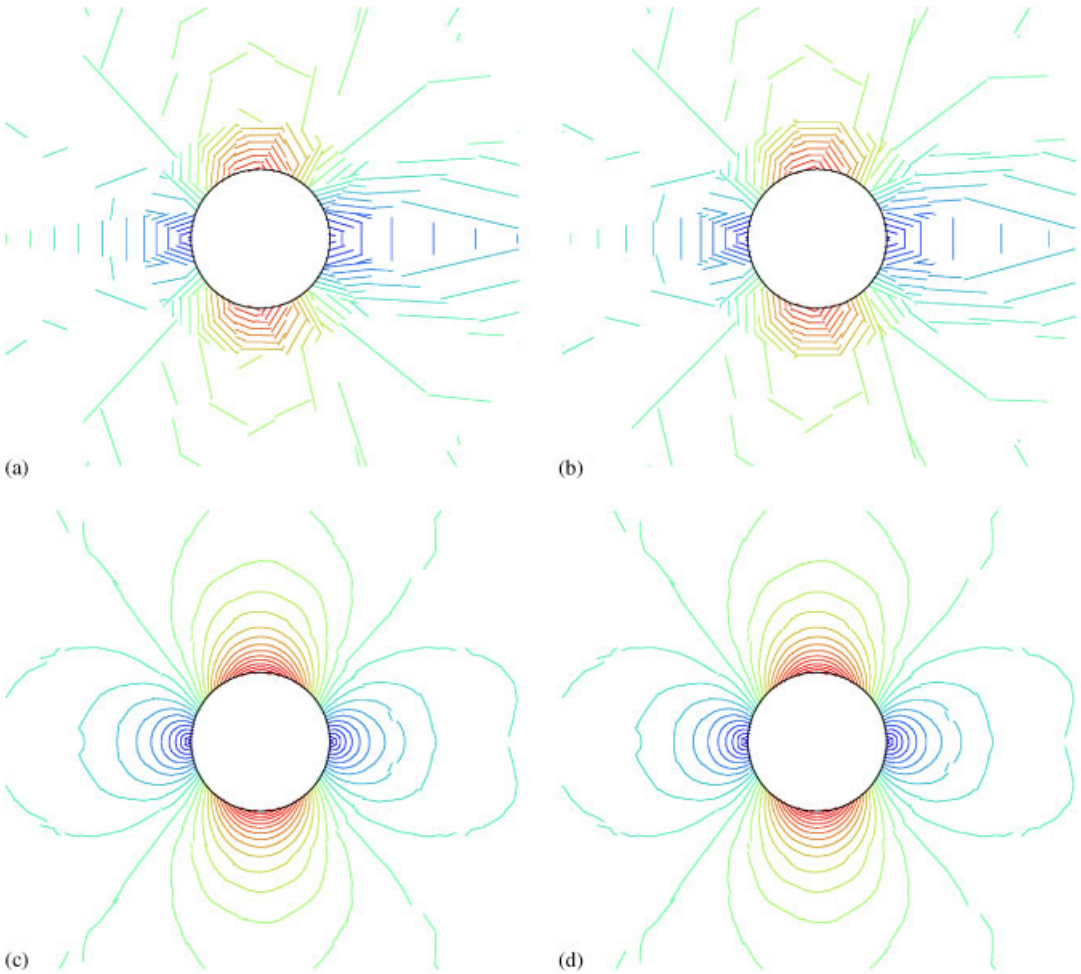

Figure 5. Mach number distributions for the flow around the circular cylinder, 16×4 mesh: (a) order 1—full quadrature; (b) order 1—quadrature free; (c) order 3—full quadrature; and (d) order 3—quadrature free.

on the coarse and the fine mesh, respectively. The strategies included in this study are:

- fully explicit v-cycle (labelled 'v-cycle');
- fully explicit w-cycle (labelled 'w-cycle');
- v-cycle with implicit coarsest level (labelled 'v-cycle-i0');
- w-cycle with implicit coarsest level (labelled 'w-cycle-i0');
- Runge–Kutta pseudo-timestepping (labelled 'Runge–Kutta');
- Newton–Krylov-ILU (labelled 'Newton–Krylov').

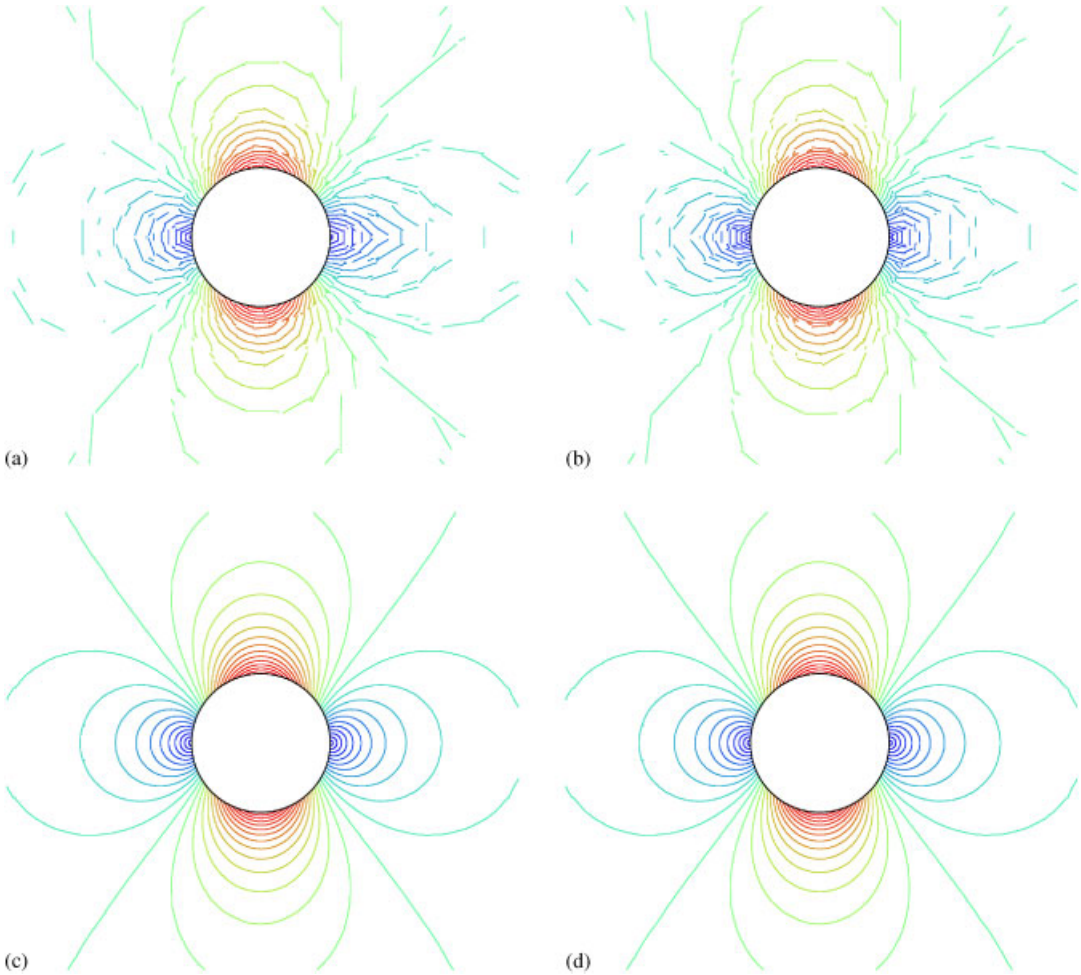

Figure 6. Mach number distributions for the flow around the circular cylinder, 32×8 mesh: (a) order 1—full quadrature; (b) order 1—quadrature free; (c) order 3—full quadrature; and (d) order 3—quadrature free.

All explicit levels use 10 Runge–Kutta pseudo-time iterations for both pre- and post-smoothing sweeps. On the implicit level we specify $\mathrm{CFL}_0 = 5$ and $\alpha = 0.5$, whilst 30 Krylov vectors are used. The Runge–Kutta pseudo-time integration and the Newton–Krylov implicit scheme both use the same parameters as their pendants on the multigrid levels.

We see that the v-cycles have the same rate of convergence per cycle irrespective of the finest-level interpolation order, i.e. we attain textbook multigrid efficiency. We see this behaviour only for v-cycles since the number of passes on each level of this strategy is independent of the finest-level interpolation order. The number of cycles however is very dependent on the grid size. This is to be expected, since the maximum stable timestep is

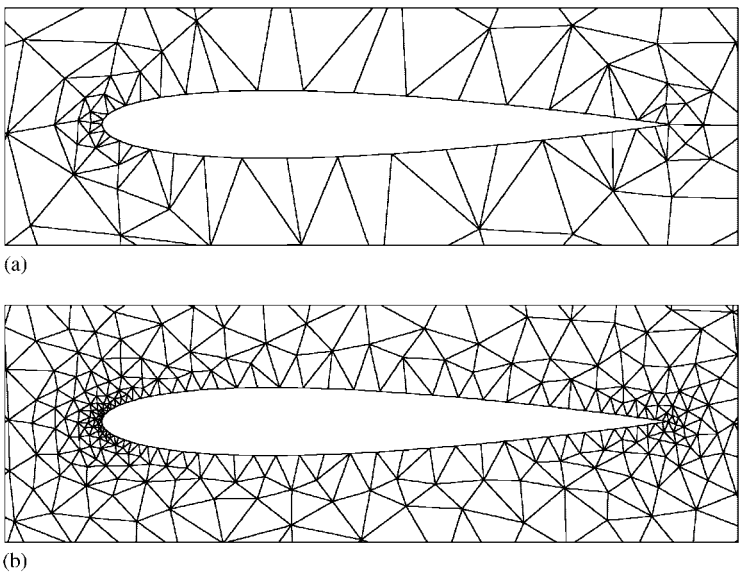

Figure 7. Meshes for the flow around the NACA0012 airfoil: (a) coarse resolution; and (b) fine resolution.

proportional to the grid size. If we had used a classical multigrid method, using coarser grids instead of decreasing order, we would have convergence rates per cycle independent of mesh size, provided the coarsest mesh remains the same. A combination of p- and classical multigrid could provide both order and grid size independent convergence rates.

We note that in any configuration (fully or partly explicit) w-cycles have a higher asymptotic convergence rate per cycle, almost in a constant ratio to the number of coarser level sweeps (v-cycle orders 2 and 3: 1 pass, w-cycle order 2: 2 passes, w-cycle order 3: 4 passes). Since the extra work is confined to the coarser levels we can sometimes see a better performance in terms of CPU time.

The application of the Newton–Krylov strategy as the smoother on the coarsest level tends to speed up v-cycles considerably, both in number of cycles and CPU time. For w-cycles the effect is most marked in the early stages, where the convergence rate seems to be dominated by low frequency errors. After an initial transient however we find asymptotic convergence rates that are similar to the ones for the fully explicit cycles. A possible explanation is that w-cycles, as they concentrate most of the workload on the coarser levels, smooth out the low-frequency error more effectively. Consequently this strategy, especially in the partly implicit case, would be dominated by the convergence of higher frequencies only in the last stage of convergence. v-cycles would tend to keep a balanced 'broad-band' error, and maintain efficiency on all levels up to the end. This effect is probably only noticeable due to the high accuracy of the transfer operators.

Finally we also note that in all cases the implicit Newton–Krylov scheme converges first. However as order and number of DOFs increases, the multigrid scheme becomes a viable

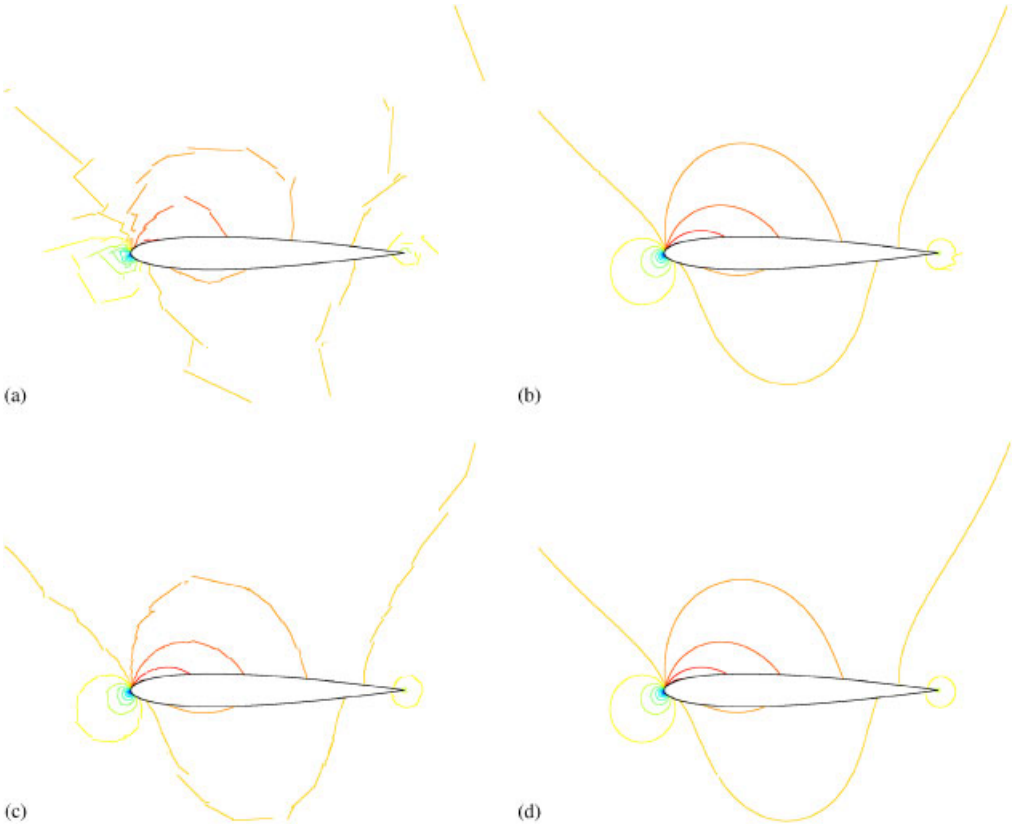

Figure 8. Mach number distribution: (a) $p=1$, coarse; (b) $p=4$ coarse; (c) $p=1$, fine; and (d) $p=4$, fine.

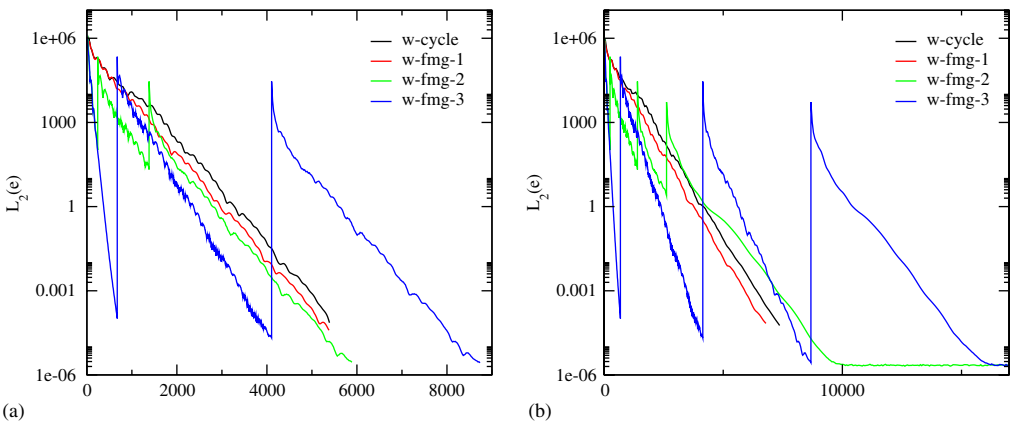

Figure 9. Comparison of FMG strategies, fine mesh: (a) order $=2$, fine mesh; and (b) order $=3$, fine mesh.

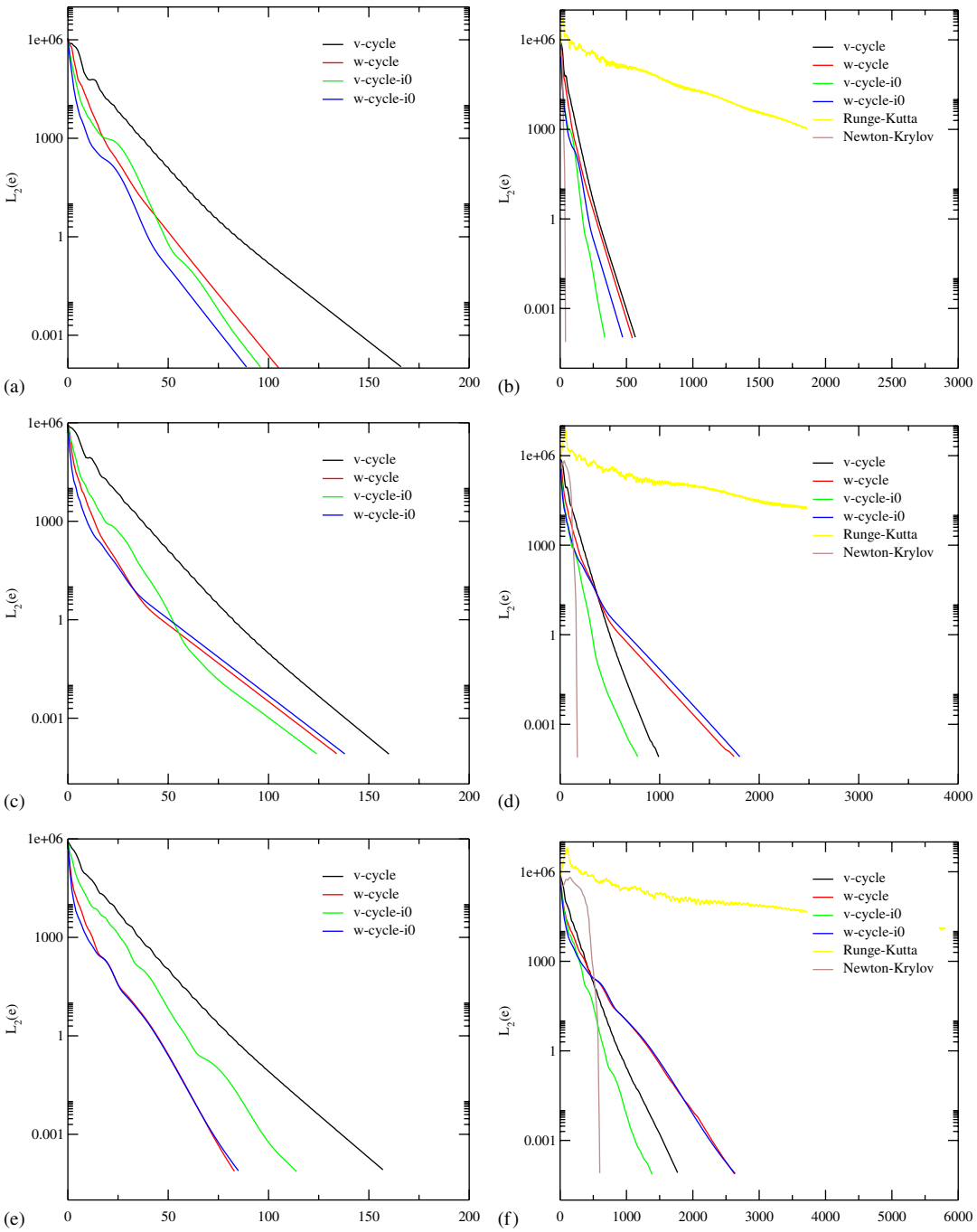

Figure 10. Comparison of iteration strategies, coarse mesh: (a) number of cycles, order = 2; (b) CPU time, order = 2; (c) number of cycles, order = 3; (d) CPU time, order = 3; (e) number of cycles, order = 4; and (f) CPU time, order = 4.

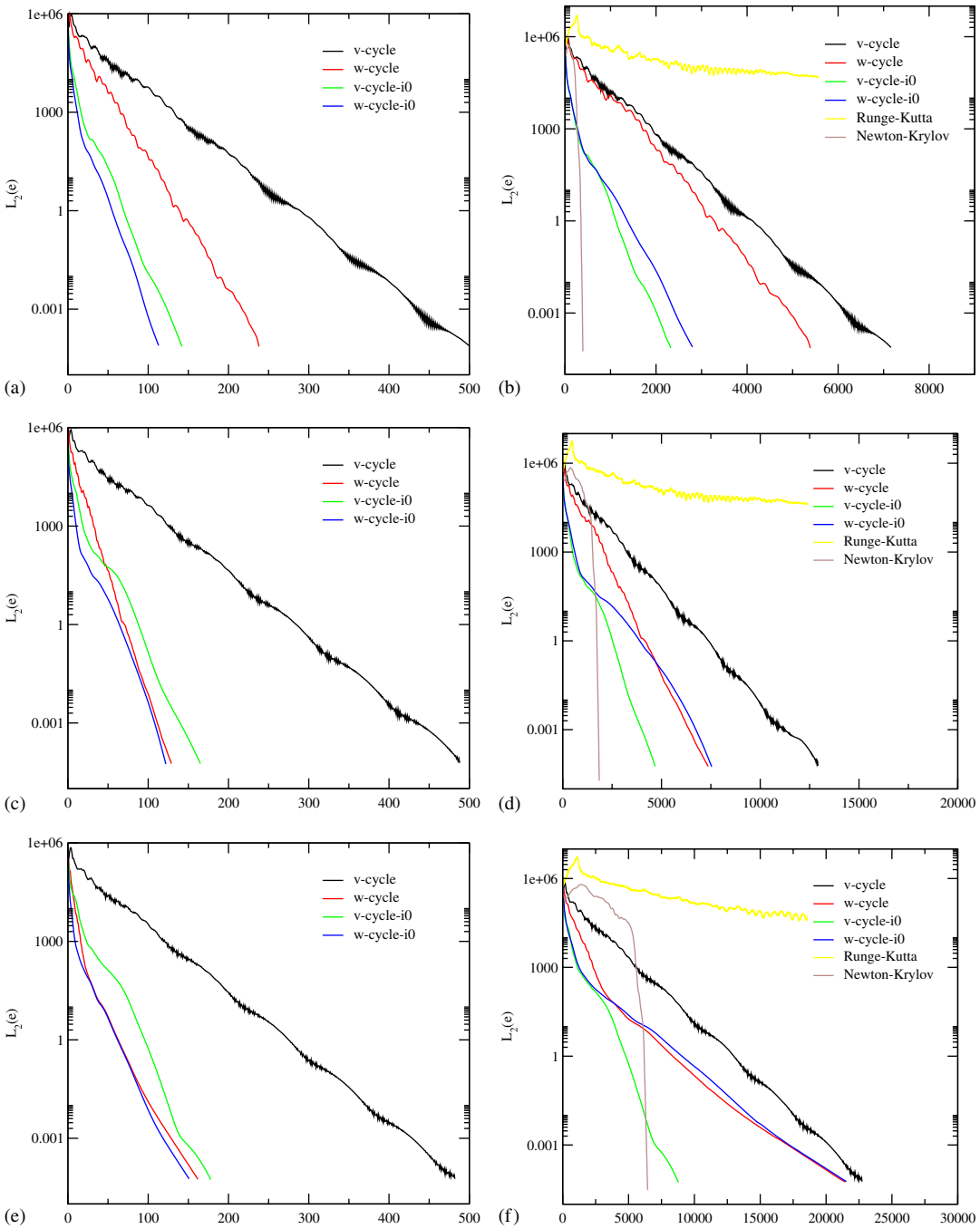

Figure 11. Comparison of iteration strategies, fine mesh: (a) number of cycles, order = 2; (b) CPU time, order = 2; (c) number of cycles, order = 3; (d) CPU time, order = 3; (e) number of cycles, order = 4; and (f) CPU time, order = 4.

competitor, even when we use a poor smoother such as Runge–Kutta, and this for a fraction of the memory.

6. CONCLUSIONS

In order to enhance the computational efficiency of DGFEM two algorithmical simplifications have been implemented. The first one is a simplified representation of the boundary, thus avoiding the necessity of isoparametric elements. The second one is a simple implementation of the quadrature free integration method. Both simplifications, although leading apparently to a decreased formal order of accuracy, retain sufficient absolute precision. The quadrature free implementation has been shown to need stabilization, and a suitable strategy has been implemented for subsonic flows. A Newton–Krylov and different p-multigrid iteration strategies have been presented and compared. Even using very simple smoothers, the p-multigrid proves to be a performant option in terms of CPU time whilst drastically reducing memory footprint and code complexity.

ACKNOWLEDGEMENTS

The visualizations and most of the meshes have been generated with the open source grid generator Gmsh (http://www.geuz.org/gmsh).

REFERENCES

1. Fidkowski KJ, Darmofal DL. Development of a higher-order solver for aerodynamic computations. *42nd AIAA Aerospace Sciences Meeting and Exhibit. AIAA Paper No. 2004-0436*, 2004.
2. Roe PL. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics* 1981; **43**:357–372.
3. Bassi F, Rebay S. High-order accurate discontinuous Galerkin finite element solution of the 2D Euler equations. *Journal of Computational Physics* 1997; **138**:251–285.
4. Baumann CE, Oden JT. An adaptive-order discontinuous Galerkin method for the solution of the Euler equations of gas dynamics. *International Journal for Numerical Methods in Engineering* 2000; **47**:61–73.
5. Krivodonova L, Berger M. High-order accurate implementation of solid wall boundary conditions in curved geometries. *Journal of Computational Physics* 2005; **211**(2):492–512.
6. Lockard DP, Atkins HL. Efficient implementations of the quadrature-free discontinuous Galerkin method. *14th AIAA CFD Conference. AIAA Paper 99-3309*, July 1999.
7. Atkins HL, Shu CW. Quadrature-free implementation of discontinuous Galerkin method for hyperbolic equations. *AIAA Journal* 1998; **36**:775–782.
8. Geuzaine P. Newton–Krylov strategy for compressible turbulent flows on unstructured meshes. *AIAA Journal* 2001; **39**(3):528–531.
9. Brown PN, Saad Y. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing* 1990; **11**(3):450–481.
10. Keyes DE, Venkatakrishnan V. Newton–Krylov–Schwarz methods: interfacing sparse linear solvers with nonlinear applications. *Zeitschrift für Angewandte Mathematik und Mechanik* 1996; **76**:147–150.
11. Rasetarinera P, Hussaini MY. An efficient implicit discontinuous spectral Galerkin method. *Journal of Computational Physics* 2001; **172**:718–738.
12. Bassi F, Rebay S. GMRES discontinuous Galerkin solution of the compressible Navier–Stokes equations. *Discontinuous Galerkin Methods*: *Theory*, *Computation and Applications*. Lecture Notes in Computational Science and Engineering, vol. 11. Springer: Berlin, 2000.
13. Cockburn B, Shu C-W. The Runge–Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems. *Journal of Computational Physics* 1998; **141**:199–224.
14. Bastian P, Reichenberger V. Multigrid for higher order discontinuous Galerkin finite elements applied to groundwater flow. *Technical Report SFB 359*, Interdisziplinäres zentrum für Wissenschaftliches Rechnen, Universität Heidelberg, 2000.
15. Wesseling G. *An Introduction to Multigrid Methods*. Wiley: New York, 1991.
16. Ladson CL, Brooks CW, Hill AS, Sproles DW. Computer program to obtain ordinates for NACA airfoils. *NASA Technical Memorandum 4741*, December 1996.

