Iterative Solvers for Discretized Stationary Euler Equations

Bernhard Pollul and Arnold Reusken

Abstract In this paper we treat subjects which are relevant in the context of iterative methods in implicit time integration for compressible flow simulations. We present a novel renumbering technique, some strategies for choosing the time step in the implicit time integration, and a novel implementation of a matrix-free evaluation for matrix-vector products. For the linearized compressible Euler equations, we present various comparative studies within the QUADFLOW package concerning preconditioning techniques, ordering methods, time stepping strategies, and different implementations of the matrix-vector product. The main goal is to improve efficiency and robustness of the iterative method used in the flow solver.

1 Introduction

Large sparse non-linear system of equations resulting from a finite volume discretization of compressible Euler equations are considered in this paper. This discretization is based on adaptive wavelet techniques for local grid refinement, For an overview of the adaptivity concept and the finite volume discretization we refer to [8]. The methods are implemented in the QUADFLOW solver, cf. [7, 8]. In this paper we only consider iterative methods for solving the large non-linear systems of equations.

The approach of a "pseudo-transient continuation" is followed. That is, an implicit time integration method is applied to the unsteady Euler equations so that the corresponding non-stationary solution converges to the stationary solution for time tending to infinity. This then yields a non-linear system of equations in each

Bernhard Pollul
Chair for Numerical Mathematics e-mail: pollul@igpm.rwth-aachen.de

Arnold Reusken
Chair for Numerical Mathematics e-mail: reusken@igpm.rwth-aachen.de

time step which is solved by a Newton-Krylov method. Therein one applies a linearization technique combined with a preconditioned Krylov subspace algorithm for solving the resulting linear problems. The computational work needed for solving the large sparse systems in the Newton-Krylov method determines to a large extent the total computing time.

Because preconditioning is crucial for the convergence of the Krylov solver we investigate different so-called "point-block" preconditioners. Preconditioners usually strongly depend on the ordering of the (block) unknowns. We present a new renumbering technique that is based on a reduced matrix graph and that can significantly improve both the robustness and the efficiency of the iterative method. The selection of the time step size in the implicit time integration is crucial for the performance of the iterative solver. We investigate two known and two novel time step selection strategies. A further acceleration of the time integration can be achieved by the use of a second order accurate Jacobian. Because the stencils for second order methods are relatively large resulting in a complex Jacobian requiring much memory, we present a second order matrix-free method that uses automatic differentiation. Using the second order matrix-free evaluation of the matrix-vector product the corresponding computational time can be significantly decreased.

2 The Euler Equations

Derived from the fundamental conservation laws of fluid dynamics, the time-dependent Euler equations describe the motion for an inviscid, non-heat-conducting compressible gas. For an arbitrary control volume $V \subset \Omega \subset \mathbb{R}^d$ with boundary ∂V and outward unit normal vector $\mathbf{n}$ on the surface element $dS \subset \partial V$ they are given by

$$\int_V \frac{\partial \mathbf{u}}{\partial t}\, dV + \oint_{\partial V} \mathbf{F}(\mathbf{u})\mathbf{n}\, dS = 0\,. \tag{1}$$

The convective flux $\mathbf{F}(\mathbf{u})$ and the vector of unknown conserved quantities $\mathbf{u}$ containing the density ρ, the static pressure p, the velocity vector of the fluid $\mathbf{v}$, and the total energy E are given by

$$\mathbf{F}(\mathbf{u}) = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\circ\mathbf{v} + p\mathbf{I} \\ E\mathbf{v} + p\mathbf{v} \end{pmatrix} \quad , \qquad \mathbf{u} = \begin{pmatrix} \rho \\ \rho\mathbf{v} \\ E \end{pmatrix}\,, \tag{2}$$

where $\circ$ denotes the dyadic product. The system is closed by suitable initial and boundary conditions and the equation of state for a perfect gas using the ratio of specific heats $\gamma = \frac{c_p}{c_v}$:

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho\mathbf{v}^2\,. \tag{3}$$

The equations (1) – (3) form the standard model that is considered in this paper.

3 Test Problems

We describe two classes of test problems which are used in the numerical experiments below.

3.1 Homogeneous Stationary Flow on the Unit Square

In this simple test problem we consider $\Omega = [0,1]^2$ on a uniform mesh $\Omega_h = \{(ih, jh) \mid 0 \le i, j \le n\}$, with $nh = 1$ and boundary conditions such that the stationary Euler equations have a constant solution. We apply the Van Leer flux vector-splitting scheme and use compatibility relations for the discretization of the boundary conditions. A lexicographic ordering of the grid points is applied. The discretization yields a non-linear system of equations

$$\mathbf{F} : \mathbb{R}^{4N} \to \mathbb{R}^{4N}, \quad \mathbf{F}(\mathbf{U}) = 0 . \tag{4}$$

The continuous constant solution (restricted to the grid) solves the discrete problem and thus the solution of the non-linear discrete problem, denoted by $\mathbf{U}^*$, is known a-priori. In Subsection 4.2 we investigate the behavior of different preconditioners when applied to a linear system of the form $DF(\mathbf{U}^*)\mathbf{x} = \mathbf{b}$. The matrix $DF(\mathbf{U}^*)$ has a *point-block* structure, that is, a regular block structure $DF(\mathbf{U}^*) = \text{blockmatrix}(A_{i,j})_{0 \le i,j \le N}$ with $A_{i,j} \in \mathbb{R}^{4\times 4}$ for all i, j. Note that $A_{i,j} \neq 0$ can occur only if $i = j$ or i and j correspond to neighboring grid points.

3.2 Stationary Flow around NACA-0012 Airfoil

This problem is a standard test case for inviscid compressible flow solvers [25] in which the inviscid, transonic, stationary flow around the NACA0012 airfoil is considered. We present results for the three test cases given in Table 1 characterized by Mach number M_∞ and angle of attack α.

NACA0012	M_∞	α
test case 2A	0.80	1.25°
test case 2B	0.95	0.00°
test case 2C	1.20	0.00°

Table 1 Test cases 2A, 2B, 2C: Mach number M_∞ and angle of attack α for NACA0012 airfoil

The problems are discretized using a hierarchy of locally refined grids on which standard finite volumes are applied, cf. [8]. The steady-state solutions of test cases

2A, 2B, and 2C are evolved in a pseudo-transient continuation solving (1), starting on a coarse initial grid, and evolving a solution on an adaptively refined grid. We perform one inexact Newton iteration per time step. The corresponding Jacobian matrices are the system matrices of the occurring systems of linear equations. These systems are solved with a left-preconditioned BiCGSTAB method. Preconditioning will be explained in detail in Section 4.

In the implicit time integration, the size of the time step is determined by a CFL number γ which is not limited by the Courant-Friedrichs-Levy (CFL) condition [16]. Initialized by γ_{MIN} on the coarse initial grid, the CFL number is increased by an evolution method as presented in Section 6 in every time step until an a-priori fixed upper bound γ_{MAX} is reached. Time integration is continued until a tolerance criterion for the residual is satisfied. Then a (local) grid refinement is performed and the procedure starts again with an initial CFL number equal to γ_{MIN}.

3.2.1 Grid Hierarchy

We show some first results for test cases 2A, 2B, and 2C, using the QUADFLOW solver with a standard time step selection strategy method ($\gamma_{k+1} = 1.1 \cdot \gamma_k$, $\gamma_{\min} = \gamma_0 = 1$, and $\gamma_{\max} = 1000$), cf. Section 6. As in [8] we allow 8 maximum levels of refinement, 10 cycles of adaptations in the cases 2A and 2C and 13 cycles in case 2B.

Test case	Grid	1	2	3	4	5	6	7
2A	# cells	400	1 384	2 947	3 805	4 636	5 689	6 817
2B	# cells	400	1 600	4 264	7 006	11 827	15 634	21 841
2C	# cells	400	1 600	4 864	10 189	16 885	23 290	30 598

Test case	Grid	8	9	10	11	12	13	14
2A	# cells	7 753	9 028	9 523	9 874	only 10 adaptations carried out		
2B	# cells	25 870	28 627	30 547	31 828	33 067	33 955	34 552
2C	# cells	36 160	38 764	39 961	40 708	only 10 adaptations carried out		

Table 2 Sequence of grids. Tabulated is the number of cells in nested grids for test cases 2A and 2C (10 adaptations performed) and for test case 2B (13 adaptations performed)

In Table 2 the sequence of nested grids for the four test cases is given. In a full simulation, the density residuals are decreased by a factor of 10^4 in the finest grid and by a factor of 10^2 on all coarser grids. Note that the finest grids contain up to 102 times as much cells as the initial grids. Therefore the focus in the following sections will be on reducing the time that is needed to achieve convergence on the finest grid. Therein, the main effort is the solution of the large, sparse linear equation systems that arise in Newton's linearization method.

4 Point-Block Preconditioners

In the Newton-Krylov approach the arising linear systems of equations are solved by a Krylov method. Therein, the choice of the preconditioner is crucial for the convergence process. Our main focus is on the incomplete LU-factorization (ILU) and Gauss-Seidel (GS) preconditioners that are widely-used in solvers in the numerical simulation of compressible flows [1, 6, 18, 31, 35, 40].

4.1 Methods

In the test problems described in the previous section we have to solve large systems of linear equations. The matrices have a point-block structure in which the blocks correspond to the $d+2$ unknowns in each of the N cells (finite volume method). Thus, we have linear systems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}\,, \quad \mathbf{A} = \text{blockmatrix}(A_{i,j})_{1\le i,j\le N}\,, \quad A_{i,j} \in \mathbb{R}^{(d+2)\times(d+2)}\,. \tag{5}$$

In the following we describe basic point-block iterative methods that are used as preconditioners in the iterative solver. For the right-hand side we use a block representation $\mathbf{b} = (b_1,\ldots,b_N)^T$, $b_i \in \mathbb{R}^{d+2}$ that corresponds to the block structure of $\mathbf{A}$. The same is done for the iterands $\mathbf{x}^k$ that approximate the solution of (5).

For the description of the preconditioners the nonzero pattern $P(\mathbf{A})$ corresponding to the point-blocks in the matrix $\mathbf{A}$ is important:

$$P(\mathbf{A}) = \{\,(i,j) \mid A_{i,j} \neq 0\,\} \tag{6}$$

4.1.1 Point-Block-Gauss-Seidel Method

The point-block-Gauss-Seidel method (PBGS) is the standard block Gauss-Seidel method applied to (5). Let $\mathbf{x}^0$ be a given starting vector. For $k \ge 0$ the iterand $\mathbf{x}^{k+1} = (x_1^{k+1},\ldots,x_N^{k+1})^T$ should satisfy

$$A_{i,i}x_i^{k+1} = b_i - \sum_{j=1}^{i-1} A_{i,j}x_j^{k+1} - \sum_{j=i+1}^{N} A_{i,j}x_j^k\,, \quad i = 1,\ldots,N\,. \tag{7}$$

This method is well-defined if the $(d+2)\times(d+2)$ linear systems in (7) are uniquely solvable, that is, if the diagonal blocks $A_{i,i}$ are nonsingular. In our applications this was always satisfied. This elementary method is very easy to implement and needs no additional storage. The algorithm is available in the PETSc library [2].

4.1.2 Point-Block-ILU(0) Method

We consider the point-block version of the standard point ILU(0) algorithm, denoted by PBILU(0). For the PBILU(0) preconditioner a preprocessing phase is needed in which the incomplete factorization is computed. Furthermore, additional storage similar to the storage requirements for the matrix $\mathbf{A}$ is needed. One can consider variants of this algorithm, e.g. PBILU(p), $p = 1,2,\ldots$. This produces additional storage requirements and additional arithmetic costs. Both, the PBILU(0) algorithm and such variants, are available in the PETSc library [2].

4.1.3 Point-Block Sparse Approximate Inverse

The SPAI method [20] can be modified to its point-block formulation in the same way as Gauss-Seidel and ILU. In its point-block version, denoted by PBSPAI(0), we use $\mathbf{M} = \text{blockmatrix}(M_{i,j})_{1\le i,j\le N}$, $M_{i,j} \in \mathbb{R}^{(d+2)\times(d+2)}$ and denote the set of admissible approximate inverses by $\mathscr{M} := \{\, \mathbf{M} \in \mathbb{R}^{(d+2)N\times(d+2)N} \mid P(\mathbf{M}) \subseteq P(\mathbf{A}) \,\}$. A sparse approximate inverse $\mathbf{M}$ is determined by minimization over this set:

$$\|\mathbf{AM} - \mathbf{I}\|_F = \min_{\tilde{\mathbf{M}}\in\mathscr{M}} \|\mathbf{A}\tilde{\mathbf{M}} - \mathbf{I}\|_F \tag{8}$$

The choice for the Frobenius norm allows a splitting of this minimization problem leading to multiple *low dimensional* least squares problems that can be solved by standard methods in *parallel*. The application of the PBSPAI(0) preconditioner requires a sparse matrix-vector product computation which also has a high parallelization potential. As for the PBILU(0) preconditioner a preprocessing phase is needed in which the PBSPAI(0) preconditioner $\mathbf{M}$ is computed. Additional storage similar to the storage requirements for the matrix $\mathbf{A}$ is needed. We also implemented the row-variant of SPAI, denoted by PBSPAI$_{row}$(0). As for the ILU preconditioner, there exist variants in which additional fill-in is allowed, cf. [20].

4.2 Numerical Experiments

We present results of numerical experiments. Our goal is to illustrate and to compare the behavior of the different preconditioners presented above for both test problems. In test problem 1, the Jacobian is evaluated at the discrete solution $\mathbf{U}^*$. The solution is trivial, namely, constant. The solution is a complex flow field in test problem 2. In the latter linear systems with matrices as in (5) arise in the solver used in the QUADFLOW package.

In all experiments below we use a left preconditioned BiCGSTAB method. For test problem 1, the discretization routines, methods for the construction of the Jacobian matrices and the preconditioners (PBGS, PBILU(0) and PBSPAI(0)) are imple-

mented in MATLAB. For the other test problems the approximate Jacobian matrices are computed in QUADFLOW using PETSc [2]. More results are presented in [32].

4.2.1 Arithmetic Costs

To measure the quality of the preconditioners we present the number of iterations that is needed to satisfy a certain tolerance criterion. We briefly comment on the arithmetic work needed for the construction of the preconditioner and the arithmetic costs of one application of the preconditioner. As unit of arithmetic work we take the costs of one matrix-vector multiplication with the matrix $\mathbf{A}$, denoted by 1 matvec.

For the PBGS method we have no construction costs. The arithmetic work per application of the PBGS preconditioner is about 0.7 matvec. In our experiments the costs for constructing the PBILU(0) preconditioner are between 2 and 4 matvecs. We typically need 1.2–1.6 matvecs per application of the PBILU(0) preconditioner. The costs for constructing the PBSPAI(0) preconditioner are much higher. Typical values (depending on $P(\mathbf{A})$) in our experiments are 20–50 matvecs. We typically need 1.2–1.5 matvecs per application of the PBSPAI(0) preconditioner.

4.2.2 Stationary 2D Euler

We consider the discretized stationary Euler equations as described in Paragraph 3.1 with mesh size $h = 0.02$. We vary the Mach number in x_1-direction, which is denoted

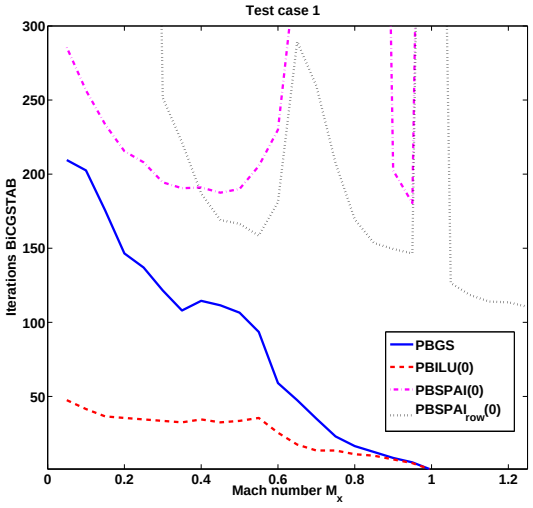

Fig. 1 Test problem 1: Iteration count for different Mach numbers

by M_x: $0.05 \leq M_x \leq 1.25$. For the Mach number in x_2-direction, denoted by M_y, we take $M_y = \frac{3}{2}M_x$. The BiCGSTAB iteration, initialized by the all-zero starting vector, is stopped if the relative residual is below 10^{-6}, measured in the 2-norm. The results are presented in Figure 1. In the supersonic case ($M_x > 1$), due to the downwind numbering, the upper block-diagonal part of the Jacobian is zero and thus both the PBILU(0) method and PBGS are exact solvers. The PBSPAI(0) preconditioner does not have this property, due to the fact that $\mathbf{M}$ is a sparse approximation of $\mathbf{A}^{-1}$, which is a *dense* block lower triangular matrix. For $M_x < 1$ with PBGS preconditioning we need about 1 to 4 times as much iterations as with PBILU(0) preconditioning. Both preconditioners show a clear tendency, namely that the convergence becomes faster if M_x is increased. For $M_x < 1$ the PBSPAI(0) preconditioners show an undesirable very irregular behavior, therefore we do not apply this preconditioner for test problem 2.

4.2.3 Stationary Flow around NACA0012 Airfoil

In the computations with the three standard NACA0012 airfoil test cases, the choice of the time step is based on an exponential strategy as already described in Paragraph 3.2.1. The linear systems with the approximate Jacobians are solved until the relative residual is smaller than 10^{-2}. In Table 3 the averaged number of preconditioned BiCGSTAB iterations for the two finest grids is given. Note that applying PBGS we need about 2–3 times as much iterations as when using PBILU(0). With PBILU(2) we save between 25% and 54% on the average iteration count compared with PBILU(0). Taking the arithmetic work per iteration into account, cf. Paragraph 4.2.1, we conclude that PBGS and PBILU(0) have comparable efficiency, whereas the PBILU(p), $p = 1, 2$, preconditioners are (much) less efficient due to the high memory requirements. In Section 5 we will see that an adequate renumbering technique significantly improves the situation for PBGS.

test case	2A		2B		2C	
Grid	10	11	13	14	10	11
PBGS	2.89	27.0	14.9	18.6	6.50	20.7
PBILU(0)	1.33	9.83	6.17	8.37	2.69	6.21
PBILU(1)	1.04	6.07	4.24	4.65	1.81	2.41
PBILU(2)	1.00	5.09	3.52	3.83	1.60	3.42

Table 3 Test problem 2: Average iteration count on two finest grids

4.3 Concluding Remarks

We summarize the main conclusions of this section. Already for our relatively simple model problems the PBSPAI(0) method has turned out to be a poor preconditioner. This method should not be used in a Newton-Krylov method for solving compressible Euler equations. Both for model problems and a realistic application (QUADFLOW solver, test problem 2) the efficiency of the PBGS preconditioner and the PBILU(0) method are comparable. For our applications the PBILU(1) and PBILU(2) preconditioners are less efficient than the PBILU(0) preconditioner.

5 Renumbering Techniques

In this section we present ordering algorithms for the PBGS preconditioner. We do not know of any literature in the context of linearized Euler equations dealing with ordering techniques for Gauss-Seidel preconditioners. The presented ordering algorithms consist of three steps. In the first step we construct a weighted directed graph in which every vertex corresponds to a block unknown and the weights correspond to the magnitude of the fluxes. This graph is usually very complex making it almost impossible to work with standard ordering techniques. Therefore, we use an approach that is very similar to coarsening techniques used in algebraic multigrid methods [37]: At first we reduce the complex graph by deleting edges with relatively small weights. Then we consider three different algorithms to determine the renumbering of the vertices of the reduced graph.

5.1 Methods

ILU and Gauss-Seidel preconditioners depend on the ordering of the cells [5, 21, 38]. This holds for their point-block variants, too. Many studies on numbering techniques for ILU preconditioners appear in the literature, cf., e.g., [18, 36] and references therein. For ILU methods, in many applications, a reverse Cuthill-McKee ordering algorithm [17] provides good results [6, 29, 34, 35]. The PBGS preconditioner can be significantly improved by reordering techniques that should be such that one approximately follows the directions in which information is propagated. In this section we introduce three renumbering methods that aim at realizing this.

All three algorithms are completely matrix-based, that is, only the block-structured matrix from (5) is needed as input. We distinguish the following three steps:

1. *Construct a weighted directed matrix graph* in which
   - every vertex corresponds to a block unknown (= cell)
   - every edge corresponds to a nonzero off-diagonal block of the given matrix **A**

2. *Construct a reduced weighted directed matrix graph* by
   • deleting edges with relatively small weights
3. *Determine a renumbering of the vertices*

While for all three algorithms presented below steps 1 and 2 are identical, they differ in the methods used in the third step. We explain the first two steps in Paragraphs 5.1.1 and 5.1.2. In Paragraphs 5.1.3 – 5.1.5 we give the three different methods that are used in step 3 to determine the reordering.

5.1.1 Construction of Weighted Directed Matrix Graph $\mathscr{G}(\mathbf{A})$

We introduce standard notation related to matrix graphs. Let $\mathscr{V} = \{1,\dots,N\}$ be a vertex set such that each vertex corresponds to a discretization cell. The set of edges $\mathscr{E}$ contains all directed edges and the mapping $\omega : \mathscr{E} \to (0,\infty)$ assigns to every directed edge $(i,j) \in \mathscr{E}$ a weight ω_{ij}:

$$\mathscr{E} = \{(i,j) \in \mathscr{V}\times\mathscr{V} \mid A_{i,j} \neq 0,\ i \neq j\} \quad , \qquad \omega_{ij} := \omega(i,j) := \|A_{i,j}\|_F \ . \tag{9}$$

We take the Frobenius-norm because it is easy to compute and all entries in a block $A_{i,j}$ are weighted equally. This yields a *weighted, directed* matrix graph $\mathscr{G}(\mathbf{A}) := (\mathscr{V},\mathscr{E},\omega)$. Opposite to the commonly used definition we call an edge $(i,j) \in \mathscr{E}$ an *inflow edge* of vertex $i \in \mathscr{V}$ and an *outflow edge* of vertex $j \in \mathscr{V}$. This is motivated by the following: In our applications, an edge (i,j) in the graph corresponds to a flow *from* cell j *into* cell i in the underlying physical problem. Consequently, for $(i,j) \in \mathscr{E}$ we call j a *predecessor* of i and i a *successor* of j. The set of predecessors of vertex $i \in \mathscr{V}$ is denoted by

$$\mathscr{I}_i := \{\, j \in \mathscr{V} \mid (i,j) \in \mathscr{E} \,\} \ . \tag{10}$$

In the construction of $\mathscr{G}(\mathbf{A})$ one only has to compute the weights ω_{ij} in (9). For storage of this information we use a sparse matrix format. Note that the size of the sparse matrix corresponding to $\mathscr{G}(\mathbf{A})$ is $N \times N$ (and not $N(d+2)\times N(d+2)$, as for $\mathbf{A}$). Hence, the costs both for the computation and for the storage of $\mathscr{G}(\mathbf{A})$ are low.

5.1.2 Construction of Reduced Matrix Graph $\hat{\mathscr{G}}$

Based on reduction techniques from algebraic multigrid methods in which *strong couplings* and *weak couplings* are distinguished [37], we separate *strong edges* from *weak edges*. For every vertex $i \in \mathscr{V}$ we neglect all inflow edges $(i,j) \in \mathscr{E}$ with a weight smaller than τ-times the average of the weights of all inflow edges of vertex i. Thus we obtain a reduced set of *strong* edges $\hat{\mathscr{E}}$ and a corresponding reduced (weighted, directed) graph $\hat{\mathscr{G}}(\mathbf{A}) := (\mathscr{V},\hat{\mathscr{E}},\omega_{|\hat{\mathscr{E}}})$:

$$\sigma_i := \frac{1}{|\mathscr{I}_i|} \sum_{j \in \mathscr{I}_i} \omega_{ij} \quad , \qquad \hat{\mathscr{E}} := \{(i,j) \in \mathscr{E} \mid \omega_{ij} \geq \tau \cdot \sigma_i\} \tag{11}$$

This simple construction of a reduced matrix graph $\hat{\mathscr{G}}(\mathbf{A})$ can be realized with low computational costs. Moreover, we can overwrite $\mathscr{G}(\mathbf{A})$ with $\hat{\mathscr{G}}(\mathbf{A})$.

The use of graph reduction is essential for the performance of the reordering techniques discussed below. Note that the parameter τ controls the size of the reduced graph: for $\tau = 0$ there is no reduction of the original graph, whereas for $\tau \to \infty$ the reduced graph contains only vertices and no edges. The choice of an appropriate value for the parameter τ is discussed in Subsection 5.2. In particular it will be shown that the performance of the numbering techniques is not very sensitive with respect to perturbations of the parameter value. We call τ "graph reduction parameter" below.

5.1.3 Downwind Numbering based on $(\mathscr{V}, \hat{\mathscr{E}})$ (Bey and Wittum)

The downwind numbering algorithm due to Bey and Wittum [5], denoted by "BW", is presented in Figure 2. This ordering is used in multigrid methods for scalar convection-diffusion problems for the construction of so-called "robust smoothers". To apply this algorithm for our class of problems we need the reduced directed graph $(\mathscr{V}, \hat{\mathscr{E}})$ as input. Note that although they have been used to compute the reduced graph $(\mathscr{V}, \hat{\mathscr{E}})$, the weights ω_{ij} are *not* used in the ordering algorithm.

```
for all P ∈ V : Index(P) := −1 ;
n_F := 1 ;
for P ∈ V
   if (Index(P) < 0 ) SetF(P) ;
end P

procedure SetF(P)
   if (all predecessors B of P have Index(B) > 0 )
      Index(P) := n_F ;
      n_F := n_F + 1 ;
      for Q successor of P
         if (Index(Q) < 0) SetF(Q) ;
      end Q
   end if
```

Fig. 2 Downwind numbering algorithm BW

Remark 1. In the loop over $P \in \mathscr{V}$ in algorithm BW the ordering of the block-unknowns (cells) corresponding to the input matrix $\mathbf{A}$ is used. In the procedure SetF(P) a vertex is assigned the next number if all its predecessors have already been numbered. Hence, the first number is assigned to a vertex that has no inflow edges. Note that in the procedure SetF(P) there is freedom in the order in which the successors Q are processed. In our implementation we again use the ordering

induced by the given matrix $\mathbf{A}$. The BW numbering is applied to the reduced matrix graph. If that graph is cycle-free, the algorithm returns a renumbering that is optimal in the sense that this reordering applied to the matrix corresponding to $\hat{\mathscr{G}}(\mathbf{A})$ results in a lower triangular matrix. However, in our problem class the reduced graphs in general contain cycles. In that case, after algorithm BW has finished, there still are vertices $P \in \mathscr{V}$ with Index(P)$= -1$, that is, there are $N - n_F > 0$ vertices that have no (new) number. The numbers $n_F, \ldots, N$ are assigned to these remaining vertices in the order induced by the input matrix ordering. The two variants of BW that are treated below in general have less of such "remaining" vertices.

Note that in the BW algorithm there are logical operations and assignments but no arithmetic operations. ◇

5.1.4 Down- and Upwind Numbering based on $(\mathscr{V}, \hat{\mathscr{E}})$ (**Hackbusch**)

```
for all P ∈ V : Index(P) := −1 ;
n_F := 1 ; n_L := N ;
for P ∈ V
   if (Index(P) < 0 ) SetF(P) ;
   if (Index(P) < 0 ) SetL(P) ;
end P

procedure SetL(P)
   if (all successors B of P have Index(B) > 0 )
      Index(P) := n_L ;
      n_L := n_L − 1 ;
      for Q predecessor of P
         if (Index(Q) < 0) SetL(Q) ;
      end Q
   end if
```

Fig. 3 Down- and upwind numbering algorithm HB

In Figure 3 we present an ordering algorithm, referred to as down- and upwind numbering and denoted by "HB", that is due to Hackbusch [21]. As input for this algorithm one needs the reduced directed graph $(\mathscr{V}, \hat{\mathscr{E}})$. The routine "SetF" is the same as in the BW algorithm in Figure 2.

Remark 2. While in the BW algorithm the vertices are ordered in one direction, namely "downwind", that is, in the direction of the flow, the algorithm due to Hackbusch uses two directions: "downwind" (SetF) and "upwind" (SetL). The computational cost of algorithm HB is comparable to that of BW. ◇

5.1.5 Weighted Reduced Graph Numbering based on $(\mathscr{V}, \hat{\mathscr{E}}, \omega_{|\hat{\mathscr{E}}})$

The performance of the BW and HB numbering depend on the ordering of the input graph. We present an algorithm that uses the weights of the reduced graph to avoid the dependence on the initial ordering. The algorithm, denoted by "WRG", is presented in Figure 4.

for all $P \in \mathscr{V}$: Index$(P) := -1$;
$n_F := 1$; $n_L := N$;

/* (i) apply SetF and SetL to starting vertices */
do in an outflow-ordered list Ⓢ, $S(\Sigma_i \omega_{ip}, P)$: for $P \in \mathscr{V}$ (12)
  if (Index$(P) < 0$) SetF$(P, 1)$;
end P
do in an inflow-ordered list Ⓢ, $S(\Sigma_j \omega_{pj}, P)$: for $P \in \mathscr{V}$ (13)
  if (Index$(P) < 0$) SetL(P) ;
end P

/* (ii) number remaining vertices */
do in an outflow-ordered list Ⓢ, $S(\Sigma_i \omega_{ip}, P)$: for $P \in \mathscr{V}$ (14)
  if (Index$(P) < 0$) SetF$(P, 0)$;
end P

procedure SetF(P, s)
  if (all predecessors B of P have Index$(B) > 0$) or $(s = 0)$
    Index$(P) := n_F$;
    $n_F := n_F + 1$;
    do in an outflow-ordered list Ⓢ, $S(\Sigma_i \omega_{iq}, Q)$: for Q successor of P (15)
      if (Index$(Q) < 0$) SetF$(Q, 1)$;
    end Q
  end if

procedure SetL(P)
  if (all successors B of P have Index$(B) > 0$)
    Index$(P) := n_L$;
    $n_L := n_L - 1$;
    do in an inflow-ordered list Ⓢ, $S(\Sigma_j \omega_{qj}, Q)$: for Q predecessor of P (16)
      if (Index$(Q) < 0$) SetL(Q) ;
    end Q
  end if

Ⓢ : p denotes the index of the vertex P of the input graph. $S(\Sigma_i \omega_{ip}, P)$ sorts the vertices P descending in the corresponding values $\Sigma_i \omega_{ip}$ (similar for $S(\Sigma_j \omega_{pj}, P)$).

Fig. 4 Weighted reduced graph numbering algorithm WRG

There are two important differences to the algorithms HB and BW. The first difference is related to the arbitrariness of the order in which the vertices are handled in the loops in HB and BW, cf. Remark 1. If there are different possibilities for which vertex is to be handled next we now use the *weights* ω_{ij} of the reduced graph to make a decision. This decision is guided by the principle that edges with larger weights are declared to be more important than those with relatively small weights. A weight based sorting occurs at several places, namely in (12) – (16). In (12) the

vertices with no inflow edges ("starting" vertices) are sorted using the sum of the weights of the outflow edges at each vertex. Similarly, in (13) the vertices with no outflow edges are sorted. The "remaining" vertices, that is, all vertices that have inflow *and* outflow edges, are finally sorted based on the sum of the outflow edges at each vertex in (14). In all three cases the number of vertices to be sorted is much smaller than N and thus the time for sorting is acceptable. Sorting is also used in (15) and (16) to determine the order in which successors and predecessors are handled. In SetF$(\cdot,\cdot)$ the successors Q of the current P are sorted using the sum over the weights of all outflow edges for each Q. This is done similarly in SetL$(\cdot)$ for all predecessors of the current P.

The second difference is that the loop over the numbering routine SetF is called *two* times. The first call SetF$(P,1)$ in part `(i)` of algorithm WRG is similar to the call of SetF(P) in the algorithms BW and HB but now with an ordering procedure used in SetF. The second call SetF$(P,0)$ (in part `(ii)` in WRG) is introduced to handle the remaining vertices that still have index value -1. In this call we do not consider the status of inflow edges and continue numbering in downwind direction (SetF(·,0)). The inner call SetF$(Q,1)$ to number the successors still requires that all predecessors have been numbered. After part `(ii)` of the algorithm is finished the only possibly not yet numbered vertices are trivial ones, in the sense that these are vertices that have no edges to other vertices.

Note that although the first part of this numbering (cf. `(i)` in Figure 4) can be also obtained by applying HB to an a-priori sorted graph, the second step `(ii)` of WRG does neither have a counterpart in HB nor in BW.

Remark 3. In all three algorithms the computational time that is needed and the storage requirements are modest compared with other components of the iterative solver. Moreover, since the Jacobian matrices of consecutive time steps are in some sense similar we apply the reordering not in each iteration but only "now and then" and keep it for the subsequent time steps, cf. Subsection 5.2. Because of the infrequent application of the numbering the total execution time for the reordering routines is very small compared with the total time needed. ◇

5.2 Numerical Experiments

We illustrate the behavior of four different numberings for a few test problems. The BW, HB and WRG methods have been explained above. The fourth numbering, denoted by QN, is induced by multiscale analysis that is used for error estimation and generates local refinement leading to a hierarchy of locally refined grids. In the QN numbering the cells are numbered level-wise from the coarsest to the finest level resulting in a sort of hierarchical block-structure of the matrix.

For efficiency reasons we do *not* apply the renumbering method (steps 1–3) to every new Jacobian but use the known renumbering as computed in the first time step. All three numbering techniques are sensitive with respect to the choice of the value for the parameter τ. In our sub- and supersonic problems $\tau = 1.25$ turned out

to be a good default value. In highly transonic problems ($M_\infty \approx 1$) the performance can often be improved by taking a somewhat large τ-value (e.g., $\tau = 2.00$).

Table 4 shows the average iteration count on the finest level for the different orderings. The average is taken over all time steps that are needed to achieve convergence on the finest discretization level for test cases 2A, 2B, and 2C. The savings compared with the QN ordering are displayed in the last rows of Table 4. In all numerical experiments the reduced matrix graph was constructed with $\tau = 1.25$. For test case 2C we give the graph $\mathscr{G}(\mathbf{A})$ and the corresponding renumbered reduced graph of a typical Jacobian matrix in Figure 5.

Test case 2A

Numbering	QN	BW	HB	WRG
Average iteration count	32.0	30.6	28.6	23.0
Saving	0%	4.4%	10.6%	28.1%

Test case 2B

Numbering	QN	BW	HB	WRG
Average iteration count	20.2	20.1	18.2	18.4
Saving	0%	0.5%	9.9%	8.9%

Test case 2C

Numbering	QN	BW	HB	WRG
Average iteration count	24.2	12.5	12.6	10.9
Saving	0%	48.3%	47.9%	55.0%

Table 4 Test problems 2A, 2B, and 2C: Average iteration count on finest level

Using the WRG renumbering method we save between 9% and 55% of PBGS-preconditioned BiCGSTAB iterations on the finest level compared with the original numbering QN. Since the renumbering has to be computed only once, the additional computational costs for WRG are negligible. The improvement is strongest for case 2C, which is due to the fact that in this case the flow is almost supersonic and thus there is a main stream in which information is transported.

Step of WRG	(i)	(ii)	Step of WRG	(i)	(ii)
$\tau \leq 0.75$	0	40 213	$\tau = 1.75$	40 207	6
$\tau = 1.00$	11 153	29 060	$\tau = 2.00$	40 211	2
$\tau = 1.25$	39 869	344	$\tau = 2.25$	40 211	2
$\tau = 1.50$	40 205	8	$\tau \geq 2.50$	40 213	0

Table 5 Test problem 2C, finest computational grid: Different values for τ. Number of cells that were numbered in steps `(i)` and `(ii)` in WRG algorithm, cf. Figure 4

For test case 2C we illustrate the dependence of the iteration count on the graph reduction parameter τ. In Figure 6 the results for $\tau = 0.25 \cdot k,\ k = 0, 1, \ldots, 12$ are

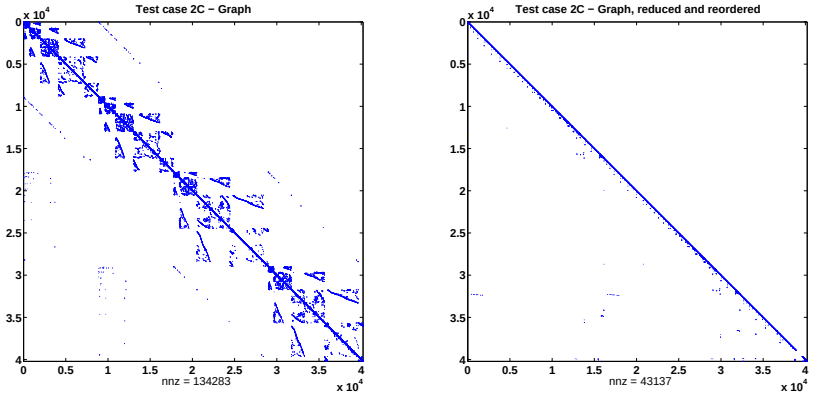

Fig. 5 Test problem 2C: Graph $\mathscr{G}(\mathbf{A})$ (left) and renumbered reduced graph (right) of Jacobian matrix on finest grid, $\tau = 1.25$.

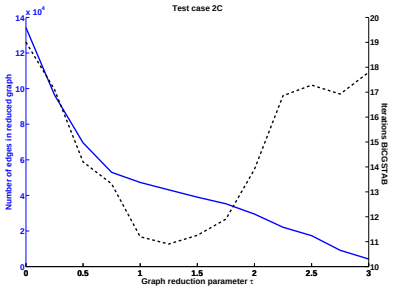

Fig. 6 Test case 2C, finest computational grid: Experiment with different values for the graph reduction parameter τ. Average iteration count using WRG numbering (dashed, right axis) and number of edges of the corresponding reduced graph (solid, left axis)

given. The dashed line (right y-axis) shows that the number of edges in the corresponding reduced graph of the Jacobian is decreasing monotonically if the value of τ is increased. Table 5 shows how many vertices are renumbered in each of the steps `(i)` and `(ii)` in the WRG algorithm, cf. Figure 4. For values $\tau \leq 0.75$ the reduced graph is too complex so that in the first step, cf. `(i)` in Figure 4, none of the 40 213 vertices is given a new number. On the other hand with $\tau = 1.50$, 98% of the vertices are given a new number in the first step `(i)`, so that a further increase of the value for τ would be counterproductive. The dashed line in Figure 6 representing the performance of the preconditioned BiCGSTAB method indicates that the choice of the value for τ is not very sensitive. For this test case values $0.75 \leq \tau \leq 2.00$ all give quite good results. We obtain the best results for $\tau = 1.25$; in this case the

reduced graph $\hat{\mathscr{G}}(\mathbf{A})$ can be reordered so that it is nearly a lower-diagonal graph as shown in the right subplot of Figure 5.

The effect of the reordering is that the dominant entries of the reordered Jacobian lie mostly in the lower triangular part. It should be noted that such reordering techniques can only be effective for point-block matrices in which there is a significant difference between $\|A_{i,j}\|_F$ and $\|A_{j,i}\|_F$ for most i, j with $\|A_{i,j}\|_F \neq 0$.

Further results are presented in [33].

5.3 Concluding Remarks

We have presented *ordering techniques for the PBGS method* that use ideas from algebraic multigrid methods. Except for the (critical) graph reduction parameter τ in (11), the ordering methods are *"black-box"*. In most test cases a good choice for this grid-reduction parameter has turned out to be $\tau = 1.25$. Only one reordering per adaptation level has been needed neglecting the additional costs of the ordering algorithm. Using the WRG reordering one can improve the *robustness* of and the *efficiency* of the linear solver.

6 Time Integration

In the pseudo-transient continuation [26], large time steps in an implicit time discretization method are preferred to achieve fast convergence. On every level of adaptation we start with an initial CFL number which determines the first time step. The local time step Δt_i for the i-th cell is given by

$$\Delta t_i = \gamma \frac{|\Omega_i|}{\lambda_i^c}\,, \quad \lambda_i^c = \oint_{\partial\Omega_i} (|\mathbf{vn}| + c)\, dS\,, \tag{17}$$

where γ is the CFL number [16] and λ_i^c is the maximum eigenvalue of the Euler equations averaged over the bounding surface of the control volume Ω_i, cf. [7].

During the time integration the CFL number is varied by one of the three strategies described in Subsection 6.1. In every time step a non-linear system of equations has to be solved. Note that the Jacobian has a structure and thus in general a smaller time step will improve the condition number of the approximated Jacobian in the Newton-Krylov method.

$$J(\mathbf{u}) = \mathrm{diag}\Big(\frac{|\Omega_i|}{\Delta t_i}\Big) + \frac{\partial \mathbf{R}(\mathbf{u})}{\partial \mathbf{u}}\,. \tag{18}$$

6.1 CFL Evolution Strategies

Implicit time integration methods in principle allow large time steps ($\gamma > 1$). For steady flows the CFL number $\gamma_k = \gamma(k)$ at a time step k is usually varied in a prescribed interval $\gamma_k \in [\gamma_{\min}, \gamma_{\max}]$. With small CFL numbers γ_k one has to perform many time steps in order to achieve convergence. Choosing the CFL number γ_k too large may result in a breakdown of the iteration process.

6.1.1 Exponential Progression (EXP)

The exponential law (EXP)

$$\gamma_{k+1} = \gamma_0 \cdot (\gamma_{\text{EXP}})^k, k \in \mathbb{N}_0 \tag{19}$$

increases the CFL number in a regular manner, also used, e.g., in [24, 39]. The control parameters γ_0 and γ_{EXP} completely determine a sequence of CFL numbers. Appropriate values for the control parameters are problem-dependent and in general not known a priori. Typical values are $\gamma_0 = \gamma_{\min}(= 1.0)$, and $\gamma_{\text{EXP}} \in [1.05, 1.5]$.

6.1.2 Switched Evolution Relaxation (SER)

The Switched Evolution Relaxation (SER) [30] method often appears in the context of compressible flow, cf., e.g., [7, 15, 27, 39]. The norm of the density residual, denoted by R_k, is directly coupled with the CFL number of the following time step:

$$\gamma_{k+1} = \gamma_{\text{SER}} \cdot \left(\frac{R_0}{R_k}\right)^{\alpha_{\text{SER}}}, k \in \mathbb{N}_0 \tag{20}$$

This way, the sequence of CFL numbers selected by the SER method is not only determined by the choice of the control parameters γ_{SER} and α_{SER} but also depends on the particular flow problem at hand. In this study we use $\alpha_{\text{SER}} \in [1.0, 5.0]$ and $\gamma_{\text{SER}} = \gamma_{\min}(= 1.0)$ as in [7]. A coupling of the CFL number with the residual is also used in the RDM strategy:

6.1.3 Residual Difference Method (RDM)

If the solutions seem to stagnate while evolving to a steady-state solution one may increase the size of the time step to check if the solution remains stable. Since in QUADFLOW the residual is taken as an estimate for the error, we suggest the following evolution strategy, referred to as Residual Difference Method (RDM), as an alternative to the SER method:

$$\gamma_{k+1} = \begin{cases} \gamma_{\min} & \text{if } k < k_0, \\ \gamma_{\mathrm{RDM}} \cdot \left(\dfrac{1}{|R_k - R_{k-1}|} \right)^{\alpha_{\mathrm{RDM}}} & \text{if } k \geq k_0, \end{cases} \qquad k \in \mathbb{N}_0 \tag{21}$$

where $k_0 \geq 1$ denotes the first index satisfying $R_{k_0} \leq R_{k_0-1} - \varepsilon_{\mathrm{RDM}}$. The control parameters are γ_{RDM} and α_{RDM}, and –like in the EXP and SER strategies– they must be selected carefully in order to obtain rapid convergenceand to avoid breakdowns. In this study, we set $\varepsilon_{\mathrm{RDM}} = 10^{-2}$ and choose $\gamma_{\mathrm{RDM}} \in \{1, 2, 5\}$ and $\alpha_{\mathrm{RDM}} \in [0.6, 6.0]$.

6.2 Numerical Experiments

We present results of numerical experiments using the different CFL evolution strategies described in Subsection 6.1, where the CFL numbers are allowed to vary in the interval $[\gamma_{\min}, \gamma_{\max}] = [1, 10^5]$. We also investigate a test case mimicking a non-adaptive scheme, called 2_nA, in which the calculation on the finest adaptation level of test case 2A is initialized with free instream conditions rather than an interpolated solution of the previous adaptation level.

The point-block-ILU(0) method is the preconditioner chosen for all tests in this section. More results are presented in [10].

6.2.1 Parameter Study on the CFL Control Parameters

We present the results of a parameter study on the CFL control parameters for the three evolution strategies. Here, the parameters γ_{EXP}, α_{SER}, α_{RDM}, and γ_{RDM} are varied, while γ_0 and γ_{SER} are assigned the fixed value 1.0.

The results for test case 2A are displayed in the upper tabular of Table 6, showing, for varying parameter values, the number of time steps needed to achieve convergence, denoted by "# ts", as well as the actual CPU time, given in seconds. All timing results are obtained on an Intel Xeon processor running at 3 GHz clock speed. It turns out that, for all three CFL evolution strategies, the choice of the control parameters has a great impact on the number of time steps needed for convergence and the total execution time. For each CFL evolution strategy, the "best" parameter values are given in bold. Results for test cases 2B and 2C are given in the other tables of Table 6. The results show that using other than the "best" values for the control parameters may lead to a doubling of time steps needed and corresponding CPU time. Because the results are equally efficient for all methods, for these test cases, the choice of the CFL evolution strategy is not the crucial factor.

Test Case 2A

EXP			SER			RDM			
γ_{EXP}	# ts	CPU	α_{SER}	# ts	CPU	α_{RDM}	γ_{RDM}	# ts	CPU
1.1	124	80.8	1.1	172	121.7	1.0	1	90	67.2
10	40	36.0	4.0	39	32.9	3.0	1	42	34.5
15	37	**31.6**	**4.5**	37	**27.2**	4.0	1	39	29.3
20	38	32.4	**5.0**	**34**	28.5	**5.0**	**1**	**36**	**28.4**
50	**35**	33.4	10.0	†	–	6.0	1	36	29.0

Test Case 2B

EXP			SER			RDM			
γ_{EXP}	# ts	CPU	α_{SER}	# ts	CPU	α_{RDM}	γ_{RDM}	# ts	CPU
1.1	115	204.1	1.1	271	406.2	1.0	1	117	216.5
3.0	56	123.7	3.0	58	135.7	2.0	5	51	112.9
5.0	46	**101.9**	4.0	51	117.8	**3.0**	**1**	**49**	113.7
10	49	115.3	**4.5**	**41**	105.6	5.0	1	50	112.3
15	**41**	125.3	**5.0**	47	**105.1**	**6.0**	**1**	50	**111.7**

Test Case 2C

EXP			SER			RDM			
γ_{EXP}	# ts	CPU	α_{SER}	# ts	CPU	α_{RDM}	γ_{RDM}	# ts	CPU
1.1	125	292.8	1.1	120	285.5	1.0	1	85	237.7
10	70	211.4	3.5	71	211.5	3.0	1	71	208.3
15	**69**	**204.8**	4.0	70	209.8	**4.0**	**1**	**70**	**205.5**
20	69	207.8	5.0	69	211.1	5.0	1	70	210.5
100	69	212.9	**10.0**	**68**	**199.1**	10.0	1	70	208.7

Table 6 Test cases 2A, 2B, and 2C: Time steps needed for convergence on finest grid and corresponding CPU times in seconds for different values for the control parameters

6.2.2 Results Mimicking a Non-Adaptive Scheme

The situation is, however, different in test case 2_nA. The results of a corresponding parameter study are presented in Table 7. It seems more difficult to find feasible

EXP			SER			RDM			
γ_{EXP}	# ts	CPU	α_{SER}	# ts	CPU	α_{RDM}	γ_{RDM}	# ts	CPU
1.05	189	116.3	2.6	1354	322.6	0.80	1	290	151.0
1.08	134	93.0	2.7	†	–	0.94	1	186	120.8
1.09	**124**	**87.6**	**2.8**	1329	**314.0**	0.95	1	†	–
1.10	†	–	**2.9**	**1320**	314.3	**0.98**	**1**	**168**	**100.5**
1.11	†	–	3.0	†	–	1.00	1	†	–

Table 7 Test case 2_nA: Time steps needed for convergence on finest grid and corresponding CPU times in seconds.

values for the control parameters because values assumed to be appropriate for test case 2A may not even yield a converging iteration process. Such a divergence in the iteration process is indicated by "†" and "–" in the table. Although EXP with $\gamma_{\text{EXP}} = 1.09$ yields the fastest convergence process, when increasing the parameter γ_{EXP} to 1.1 or higher the iteration process diverges. Compared to EXP, the SER strategy leads to a much slower convergence. The reason for this slow convergence of SER is that it chooses only relatively small CFL numbers γ_k in the first 1200 iterations. For a feasible pair of values, RDM is significantly faster than SER.

We conclude from this experiment that SER is not suitable for non-adaptive schemes. Therefore, we rather recommend using an exponential law (EXP) or RDM.

6.3 Locally Optimal CFL Numbers

Since there is no clear winner among the three basic strategies, a different approach is presented in this subsection. Reconsider that the relative density residual is used in the stopping criterion in QUADFLOW. For each time step k we define a function

$$R_k : \mathbb{R}_+ \to \mathbb{R}_+, \quad \gamma_k \mapsto R_k(\gamma_k) \tag{22}$$

which maps a CFL number γ_k to the norm of the density residual R_k that is obtained after performing this iteration using γ_k. Some typical plots of the function (22) are given in Figure 7. Apparently the shape of the functions does not change much from one time step to the following time step.

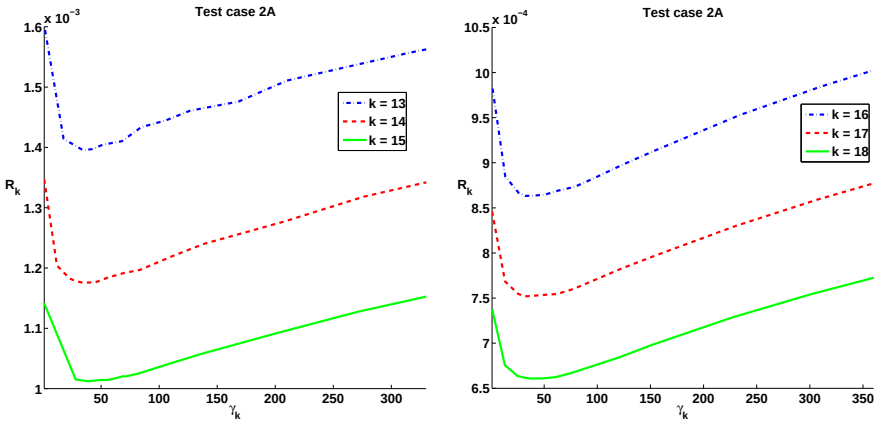

Fig. 7 Relative residual of density R_k for different values γ_k for test case 2A on the finest grid. The residuals R_k for time steps $k = 13$ to $k = 15$ and for $k = 16$ to $k = 18$ are shown in the left and right subplot, respectively. The CFL numbers $\gamma_1, \ldots, \gamma_{k-1}$ for the first $k-1$ time steps are selected by the LOC strategy, that is, by approximating (23).

The idea is to find the best CFL number in every iteration, that is, we determine a value for γ_k such that the residual gets as small as possible:

$$R_k(\gamma_k) = \min_{\gamma \in \mathbb{R}_+} R_k(\gamma). \tag{23}$$

Note that the residual R_k in the k-th iteration depends not only on γ_k, but also on the CFL numbers $\gamma_1, \ldots, \gamma_{k-1}$ used in the previous iterations.

In order to test this approach, we implemented a heuristic search strategy approximating γ_k, denoted by LOC in the sequel, where in each iteration several trial steps using different values for CFLare carried out. In the neighborhood of the CFL value yielding a minimal residual further trial steps are performed. From the set of CFL numbers tested during this heuristic search, the best CFL number, that is, the value γ_k that yields the smallest residual, is then employed to perform the actual iteration. This method is (very) expensive and therefore only of theoretical concern. This approach can be used in a faster strategy using derivative information of the function (22) so that no additional trial steps have to be carried out, cf. Remark 4. The picked CFL numbers γ_k to actually perform time step k is the one that corresponds to the smallest residual R_k. As indicated in Figure 7, for time steps $13-18$ a clear decrease of the residuals R_k can be observed in every time step.

However, it turns out that this method does not necessarily decrease the total number of iterations. In fact, the total number of iterations needed for convergence is typically larger than if any of the other methods described in section Subsection 6.1 were used. Results for the "pure" EXP strategy and switches after $n_{\mathrm{LOC}} - 1$ time steps to the LOC strategy, denoted by LOC(n_{LOC}), reveal that, as soon as the LOC strategy is initiated, the relative density residual decreases quite fast. In subsequent iterations, the rate of decrease of the residual gets smaller such that almost no progress can be observed. In the long run, the pure EXP strategy yields faster convergence although the residual actually *increases* during several iterations. This result is presented in Figure 8.

Remark 4. In the selected example, test case 2A, a closer look at the convergence behavior corresponding to Figure 8 shows that the position of the shock is slightly moving during the time integration. This can be interpreted as a "coarse grid effect". Avoiding this effect can be achieved by a more precise solution on the previous grid leading to a better initial solution on the finest grid. A decrease of the tolerance on the next coarser grid to $\varepsilon_1 = 10^{-3}$ yields a fast convergence for the LOC(2) strategy, denoted by LOC, for test case 2A. Because the LOC strategy is very expensive, we compare the results not only with the EXP strategy but also with an approximation of the LOC strategy, denoted by ADL. This approximation is feasible thanks to the similar shapes of the functions (22), cf. Figure 7. The ADL strategy uses two derivatives of the function (22) obtained by automatic differentiation and approximates $\gamma_k \mapsto R_k(\gamma_k)$ by a quadratic polynomial.

This approach works fine in some of the test cases, however, it does not perform satisfactory for test case 2C. Figure 9 shows the residual history for the three strategies for test cases 2A (left subplot) and 2C (right subplot). ◇

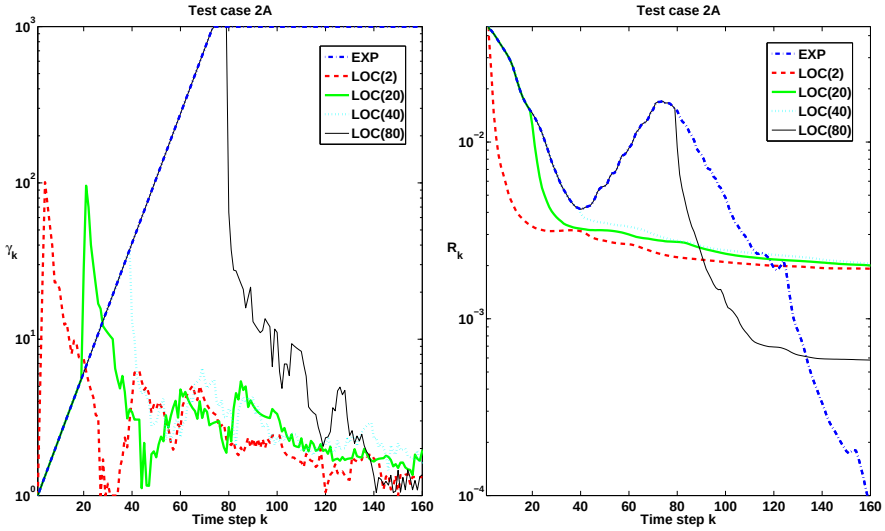

Fig. 8 Test case 2A: CFL numbers γ_k selected by EXP and LOC(n_{LOC}), for various n_{LOC} (left), and corresponding residual history R_k (right)

Remark 5. Reconsidering the right subplot of Figure 8, it seems that a fast iteration process must allow an increase of the residual R_k. We have noted in Paragraph 6.2.2 that this fact results in a slow convergence behavior when using the SER method. A similar effect eventually yields very small CFL numbers when using the LOC(n_{LOC})

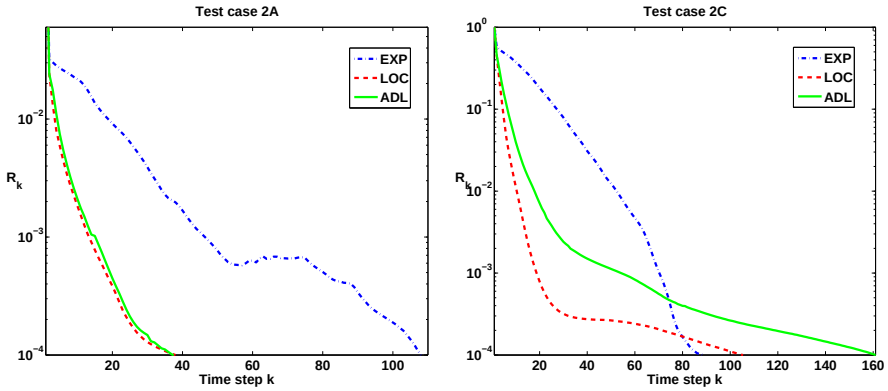

Fig. 9 Test cases 2A (left) and 2C (right): Residual histories on finest grid when solving on coarser levels to the smaller tolerance $\varepsilon_1 = 10^{-3}$, EXP, LOC, and ADL strategies

strategy. In test case 2_nA, the calculation on the finest adaptation level of test case 2A was initialized with free instream conditions. Apparently the initial residual vanishes in most of the cells. As reported in [26] an increase of the residuals is natural. Even in an adaptive computation, the same effect can occur. ◇

To better understand the impact of the CFL numbers on the residuals, a sensitivity analysis has been carried out in [10] using automatic differentiation for evaluating sensitivities without additional truncation error. The analysis has confirmed that CFL control is a subtle issue and that the three basic strategies have comparable sensitivities.

6.4 Concluding Remarks

The results have shown that the best strategy does not have to locally minimize the density residuals as much as possible in every time step, and that even an increase of the residual must be accepted in order to achieve rapid overall convergence. A new CFL evolution strategy, called RDM, has been introduced and compared with the existing strategies EXP and SER. For the residual-based strategies SER and RDM, RDM has turned out to be faster than SER. Currently, application-specific knowledge, intuition, and trial and error are still needed in order to determine appropriate values for the CFL control parameters. Using all CFL evolution strategies within an expert system like advocated in [39] may improve this situation.

7 Matrix-free Methods for Second Order Jacobians

Matrix-free evaluations of matrix-vector products are popular because the system matrix does not have to be stored and thus, one can simulate problems with larger stencils or that would not fit into memory when explicitly building the Jacobian.

The Krylov subspace method does not require the system matrix J in (18) explicitly but needs the evaluation of the product of J with some given vector $\mathbf{x} \in \mathbb{R}^{(d+2)N}$. Hence, an evaluation of the Jacobian-vector product can be realized without actually storing the Jacobian matrix.

In contrast to the approach of using divided differences to approximate the Jacobian-vector product we use AD that does not produce any additional truncation error. Thus, the derivatives can be computed more accurately than any approach using divided differences. The computational effort to compute a Jacobian-vector product by the forward mode of AD is typically similar to the computational cost of a first order divided difference approximation. For additional information on automatic differentiation and the actual implementation we refer to the article by Arno Rasch within this issue of the book.

7.1 Numerical Experiments

It is known from experience, as reported in, e.g., [29, 31], that a benefit from second order methods can only be expected after a certain number of time steps have been elapsed. Usually in the early iterations a first order implementation of a matrix-vector product is faster and more robust. In our numerical experiments we therefore usually switch at a certain threshold ν for the density residual between the different methods. A similar approach is followed in [6, 29]. In the actual implementation we do not switch back from the higher order method to the lower-order method if the residual increases again during the computation. If k_ν denotes the first time step satisfying $R_{k_\nu-1} \leq \nu$ the Jacobian approximations are as follows:

$$\left.\begin{array}{ll} J_{\text{low}}\Delta\mathbf{u} = -\mathbf{R}_{\text{high}}, & k < k_\nu \\ J_{\text{high}}\Delta\mathbf{u} = -\mathbf{R}_{\text{high}}, & k \geq k_\nu \end{array}\right\} \tag{24}$$

In the following subsection we also use a variant in which we switch at an a-priori prescribed time step k_ν. This approach is also used, e.g., in [31]. If not stated otherwise, we use the PBILU(0) preconditioner for the first order methods and a PBILU(2) preconditioner, based on the *first* order Jacobian, for the matrix-free second order method. Other results are presented in the article by Gero Schieffer within this issue of the book and in [9].

7.1.1 Newton Convergence

We investigate the impact of the CFL number on the performance of Newton's method using the first and second order methods by carrying out a fixed number of time steps using the first order matrix-based method. In this simulation the corresponding value for k_ν was chosen to be $k_\nu = 20$ (upper row of Figure 10) and $k_\nu = 80$ (lower row of Figure 10). Thereafter we perform 20 Newton steps with the matrix-based first order method (left subplots) and matrix-free second order method (right subplots). The plots show the Newton iteration history for different CFL numbers $\gamma = 10^0, 10^1, \ldots, 10^4$.

The first order method shows linear or slower than linear convergence for all tested values for γ in both time steps (cf. left plots in Figure 10). The larger the CFL number is selected, the slower the convergence of Newton's method is. This is expected because for larger time step sizes the non-linearity of the corresponding non-linear system of equation is increasing. Note that Newton's method converges only locally and the corresponding initial guesses have to be in the region for that the non-linear method converges. In time step 20 the flow is still in its startup phase, that is, most flow features (such as position of shocks) are not resolved, and thus even a a divergence of Newton's method may occur. Nevertheless, all 20 Newton iterations can be performed without a breakdown (cf. Subsection 6.1). As expected, the convergence for Newton's method is better in time step 80.

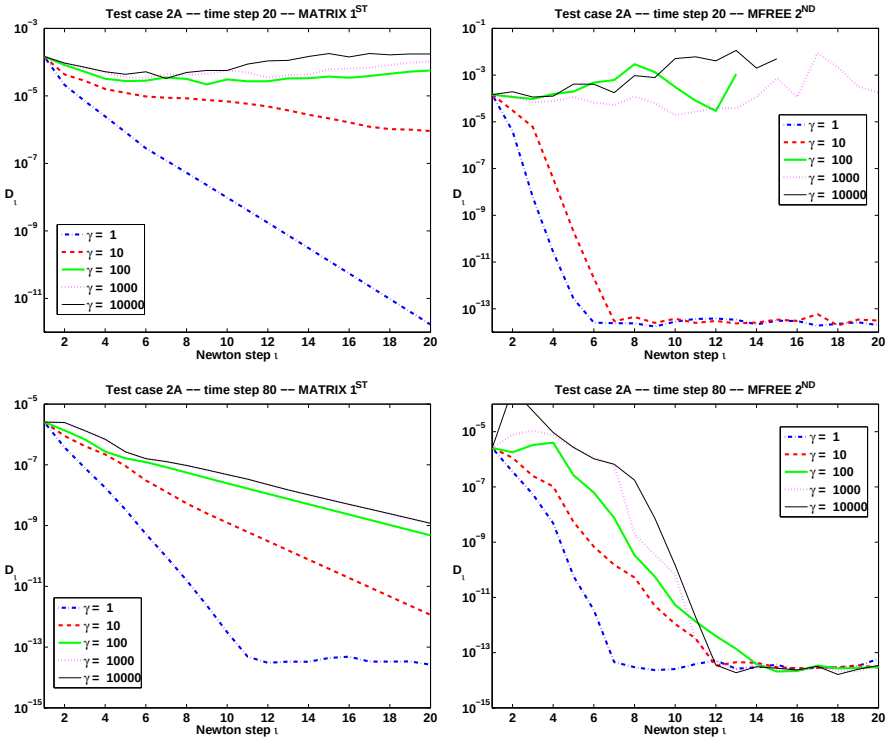

Fig. 10 Test case 2A: Newton history for different values of the CFL number γ in time steps 20 (upper plots) and 80 (lower plots) for the matrix-based first order method (left subplots) and the matrix-free second order method (right subplots)

Although the exact Newton method has the advantage of local quadratic convergence, it may be counterproductive to use it in the start-up phase of the computation. This is demonstrated by the fact that two graphs in the upper right subplot of Figure 10 show that the iteration process does not converge for some higher values for γ when using the second order method. This divergence occurs if the initial guess for Newton's method is too far away from the corresponding solution or the Krylov solver diverges when solving the corresponding linear system of equations. However, typical values at time step 20 are $\gamma \in [1,10]$ and it can be observed that for $\gamma = 1$ and $\gamma = 10$ the corresponding convergence of Newton's method is significantly faster (than for the first order method). The second order method does not show any divergence and the corresponding convergence is significantly faster in time step 80. Although the convergence of Newton's method is faster for smaller values of γ in any case —which is related to the fact that the non-linearity of (18) increases with larger time step sizes— in the computation with $k_V = 80$, the second order method shows a significant faster convergence for all —and especially

for large— CFL numbers than the first order method, as shown in the two plots in the lower row of Figure 10.

This is a major benefit of the second order method: Towards the end of the computation a significant acceleration of the time integration process can be achieved also due to the selection of larger CFL numbers γ.

7.1.2 Acceleration of Time Integration

We study the impact of ν in order to reduce the execution time of the time integration. As for all stationary test cases in the previous sections, only one Newton step is performed per time step. In Figure 11 numerical results using an exponential CFL

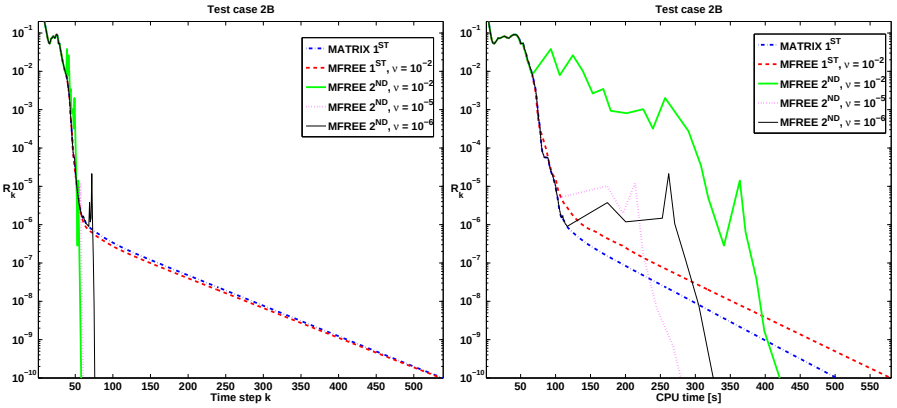

Fig. 11 Test case 2B: Residual history on finest grid in terms of iterations (left plot) and corresponding CPU time (right plot) for different values of ν for the second order matrix-free method compared with the first order methods

strategy with $\gamma_{EXP} = 1.5$, $\gamma_{MAX} = 10^6$ are presented. We plotted the residual history in terms of iteration number (left subplot) and the corresponding execution time (right subplot) for the matrix-based first order method and the second order method. We give results for different switch tolerances ν. It can be observed that the second order methods need in general less iterations. But the effect of the selection of ν is crucial: ν has to be selected small enough so that not most of the iterations are performed by the —potentially slower— first order method. On the other hand, if the switch tolerance ν is chosen *too* small the number of iterations is only insignificant smaller resulting in a bad CPU behavior. We also show a plot for the first order matrix-free variant.

7.2 Concluding Remarks

The convergence of Newton's method is significantly better for the second order method than for the first order method, especially when using larger time step sizes. This allows a clear reduction in the time steps needed for achieving convergence. However, in the test problems for the stationary two-dimensional Euler equations, a benefit from the second order method can only be achieved in the final iteration process We have shown that switching at the "right" time, i.e. $\nu = 10^{-5}$, from the first order method to the second order method can speed up the overall computation process compared with the "pure" first order matrix-based method.

8 Outlook

We conclude with some remarks on topics that could be considered in future work in this research area. The first two aspects address preconditioning, the last item suggests an expert system for time stepping and switching between first order matrix-based and second order matrix-free methods.

Parallel Preconditioners

The preconditioner and the corresponding ordering routines have to be adapted and optimized for a fully parallel version of QUADFLOW. The use of parallel ILU-type preconditioners is possible if a multi-color ordering or subdomain preconditioning is used. These approaches are investigated and compared in, e.g., [4]. For a parallel preconditioner a Newton-Krylov-Schwarz algorithm [19] using overlapping domains could be used. Such additive or multiplicative Schwarz preconditioners can be viewed as an overlapping block-Jacobi or block-Gauss-Seidel preconditioner, respectively [3]. This technique is widely-used in the context of partial differential equations as reported in many articles in the proceedings on the annually conferences on domain decomposition methods. A cheap and fast variant, the Restricted Additive Schwarz Method (RAS, RASM) by Cai and Sarkis [13], that is also integrated in the PETSc library [2], has been successfully combined with a Newton-Krylov method within the context of flow solvers [12, 19, 22, 23]. As demonstrated in [19] the RAS method can also be combined with the PBILU(0) algorithm for the subdomains. While the RAS method is used as preconditioner for the linear systems in [13], Schwarz preconditioners can also be used as a preconditioner for the non-linear systems of equations as presented in [11]. This non-linear technique was applied to a one-dimensional compressible flow, denoted by "additive Schwarz preconditioned inexact Newton method" (ASPIN), in [12] and has also been successfully used in [23]. ASPIN is a non-linear block-Jacobi iteration followed by a Newton linearization. This non-linear Schwarz preconditioner could significantly enlarge the region for that the non-linear solver converges compared with Newton's method.

Matrix-Free Preconditioners

The matrix-based preconditioner PBGS, in which J_{low} is used for the computation of

the preconditioner, can be replaced by some kind of matrix-free preconditioner in the second order matrix-free implementation of the matrix-vector product. A general approach for building a matrix-free preconditioner can be found in [14]. The use of a second order matrix-free preconditioner could certainly reduce the storage requirements for the first order matrix-based preconditioner. One could also implement a symmetric variant of PBGS, such as the matrix-free LU-SGS preconditioner which is proposed in [28]. A symmetric PBGS-type preconditioner can also be used with the described WRG ordering. In principle the renumbering technique works in a matrix-free context because only the relatively small reduced graph has to be stored.

Expert Systems for Time Integration

The implicit time integration process may be automated by some kind of advanced expert system leading to a kind of "black-box" CFL evolution strategy. A basic expert system is proposed in [39]. One can think of a complex expert system including all basic strategies, the ADL strategy, plausibility checks, a breakdown control, as well as repetitions of time steps or the use of multiple Newton steps. In a more advanced expert system different switches between the CFL evolution strategies and the first and second order methods can be realized including also switches between different preconditioners and Krylov methods.

Acknowledgment

The research for this article has been performed with funding by the Deutsche Forschungsgemeinschaft (DFG) in the Collaborative Research Center SFB 401 "Flow Modulation and Fluid-Structure Interaction at Airplane Wings" of RWTH Aachen University. We acknowledge the fruitful collaboration with several members of the QUADFLOW research group.

References

1. K. Ajmani, W.-F. Ng, and M. Liou. Preconditioned conjugate gradient methods for the Navier-Stokes equations. *Journal of Computational Physics*, 110(1):68–81, 1994.
2. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, Basel, Switzerland, 1997.
3. R. Barett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. A. van der Vorst. *Templates for the Solution of Linear Sparse Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, USA, 1994.
4. M. Benzi, W. Joubert, and G. Mateescu. Numerical experiments with parallel orderings for ILU preconditioners. *Electronic Transactions on Numerical Analysis*, 8:88–114, 1998.
5. J. Bey and G. Wittum. Downwind numbering: robust multigrid for convection-diffusion problems. *Applied Numerical Mathematics*, 23(1):177–192, 1997.

6. M. Blanco and D. W. Zingg. Fast Newton-Krylov method for unstructured grids. *AIAA Journal*, 36(4):607–612, 1998.
7. F. D. Bramkamp. *Unstructured h-Adaptive Finite-Volume Schemes for Compressible Viscous Fluid Flow*. PhD thesis, RWTH Aachen University, 2003.
8. F. D. Bramkamp, P. Lamby, and S. Müller. An adaptive multiscale finite volume solver for unsteady and steady state flow computations. *Journal of Computational Physics*, 197(2):460–490, 2004.
9. F. D. Bramkamp, B. Pollul, A. Rasch, and G. Schieffer. Matrix-free second-order methods in implicit time integration for compressible flows using automatic differentiation. Technical Report 287, IGPM, RWTH Aachen University, 2008.
10. H. M. Bücker, B. Pollul, and A. Rasch. On CFL evolution strategies for implicit upwind methods in linearized Euler equations. *International Journal for Numerical Methods in Fluids*, 59(1):1–18, 2009.
11. X.-C. Cai and D. E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM Journal on Scientific Computing*, 24(1):183–200, 2002.
12. X.-C. Cai, D. E. Keyes, and D. P. Young. A nonlinear additive Schwarz preconditioned inexact Newton method for shocked duct flows. In N. Debit, M.Garbey, R. Hoppe, J. Periaux, D. Keyes, and Y. Kuznetsov, editors, *Proceedings of the 13th International Conference on Domain Decomposition Methods, Lyon, France*, pages 345–352, 2001.
13. X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(2):792–797, 1999.
14. T. F. Chan and K. R. Jackson. Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms. *SIAM Journal on Scientific Computing*, 5(3):533–542, 1984.
15. T. Chisholm and D. W. Zingg. A fully coupled Newton-Krylov solver for turbulent aerodynamic flows. In *ICAS 2002 Congress, Toronto, ON, Canada*, Paper 333. 2002.
16. R. Courant and K. O. Friedrichs. *Supersonic Flow and Shock Waves*, volume 21 of *Applied Mathematical Sciences*. Springer, Berlin, Germany, 1999, reprint from 1948.
17. E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th national conference*, pages 157 – 172, New York, NY, USA, 1969.
18. E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM Journal on Matrix Analysis and Applications*, 13(3):944–961, 1992.
19. W. D. Gropp, D. E. Keyes, L. C. McInnes, and M. D. Tidriri. Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD. *International Journal of High Performance Computing Applications*, 14(2):102–136, 2000.
20. M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.
21. W. Hackbusch. On the feedback vertex set for a planar graph. *Computing*, 58:129–155, 1997.
22. P. D. Hovland and L. C. McInnes. Parallel simulation of compressible flow using automatic differentiation and PETSc. *Parallel Computing*, 27(4):503–519, 2001.
23. F.-N. Hwang and X.-C. Cai. A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations. *Journal of Computational Physics*, 204(2):666–691, 2005.
24. E. Issman, G. Degrez, and H. Deconinck. Implicit upwind residual-distribution Euler and Navier-Stokes solver on unstructured meshes. *AIAA Journal*, 34(10):2021–2028, 1996.
25. D. J. Jones. Reference test cases and contributors. In *AGARD–AR–211: Test Cases for Inviscid Flow Field Methods*. Advisory Group for Aerospace Research & Development, Neuilly-sur-Seine, France, 1986.
26. C. T. Kelley and D. E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis*, 35(2):508–523, 1998.
27. D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.
28. H. Luo, D. Sharov, J. D. Baum, and R. Löhner. Parallel unstructured grid GMRES+LU-SGS method for turbulent flows. *AIAA Paper 2003–0273*, 2003.

29. L. Manzano, J. V. Lassaline, P. Wong, and D. W. Zingg. A Newton-Krylov algorithm for the Euler equations using unstructured grids. *AIAA Paper 2003–0274*, 2003.
30. W. A. Mulder and B. van Leer. Experiments with implicit upwind methods for the Euler equations. *Journal of Computational Physics*, 59(2):232–246, 1985.
31. A. Nejat and C. Ollivier-Gooch. Effect of discretization order on preconditioning and convergence of a high-order unstructured Newton-GMRES solver for the Euler equations. *Journal of Computational Physics*, 227(4):2366–2386, 2008.
32. B. Pollul. Preconditioners for linearized discrete compressible Euler equations. In P. Wesseling, E. Oñate, and J. Périaux, editors, *Proceedings of the European Conference on Computational Fluid Dynamics ECCOMAS*, Egmond aan Zee, The Netherlands, 2006.
33. B. Pollul and A. Reusken. Numbering techniques for preconditioners in iterative solvers for compressible flows. *International Journal for Numerical Methods in Fluids*, 55(3):241–261, 2007.
34. A. Pueyo and D. W. Zingg. Efficient Newton-Krylov solver for aerodynamic computations. *AIAA Journal*, 36(11):1991–1997, 1998.
35. Y. Saad. Preconditioned Krylov subspace methods for CFD applications. In W. Habashi, editor, *Solution techniques for Large-Scale CFD-Problems*, pages 139–158, John Wiley & Sons, New York, NY, USA, 1995.
36. Y. Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, PA, USA, second edition, 2003.
37. K. Stüben. An introduction in algebraic multigrid. In U. Trottenberg, C. Osterlee, and A. Schüller, editors, *Multigrid*, pages 413–532, 2001. Academic Press, London, GMD Birlinghoven, St. Augustin, Germany, 2001.
38. S. Turek. On ordering strategies in a multigrid algorithm. In W. Hackbusch and G. Wittum, editors, *Notes on Numerical Fluid Mechanics*, volume 41 of *Proc. 8th GAMM-Seminar, Kiel*. Vieweg, Braunschweig, Germany, 1997.
39. D. Vanderstraeten, A. Csík, and D. Rose. An expert-system to control the CFL number of implicit upwind methods. Technical Report TM 304, Universiteit Leuven, Belgium, 2000.
40. P. Wong and D. W. Zingg. Three-dimensional aerodynamic computations on unstructured grids using a Newton-Krylov approach. *Computers & Fluids*, 37(2):107–120, 2008.