Preconditioners for linearized discrete compressible Euler equations*

Bernhard Pollul† Arnold Reusken†

September 21, 2004

Subject Classification: [msc2000] 65F10, 65N22
Key Words: Euler equations, Krylov subspace methods, point-block preconditioners.

1 Introduction

We are interested in iterative methods for solving the large sparse nonlinear systems of equations that result from the discretization of stationary compressible Euler equations. Related to the discretization it is important to distinguish two approaches. Firstly, a direct spatial discretization (using finite differences or finite volume techniques) applied to the stationary problem results in a corresponding nonlinear discrete problem. In the second approach the stationary solution is characterized as the asymptotic (i.e., time tending to infinity) solution of an evolution problem. In such a setting one applies a time integration method to the instationary Euler equations. In cases where one has very small spatial grid sizes, for example if one uses grids with strong local refinements, an implicit time integration method should be used. This then yields a nonlinear system of equations in each timestep. Note that for a given spatial grid in the first approach we have *one* discrete nonlinear problem whereas in the second approach we obtain a *sequence* of discrete nonlinear problems.
For solving such nonlinear systems of equations there are many different approaches. Here we mention two popular techniques, namely nonlinear multigrid solvers and Newton-Krylov methods. Well-known nonlinear multigrid techniques are the FAS method by Brandt [8], the nonlinear multigrid method by Hackbusch [12] and the algorithm introduced by Jameson [15]. It has been shown that a nonlinear multigrid approach can result in very efficient solvers, which can even have optimal complexity ([16, 20, 21, 23]). For these methods, however, a coarse-to-fine grid hierarchy must be available. The Newton-Krylov algorithms do not require this. In these methods one applies a linearization technique combined with a preconditioned Krylov subspace algorithm for solving the resulting linear problems. One then only needs the system matrix and hence these methods are in general much easier to implement than multigrid solvers. Moreover, one can use efficient implementations of templates that are available in sparse matrix libraries. Due to these attractive properties the Newton-Krylov technique is also often used in practice (cf., for example, [22, 24, 25, 26, 27, 35]).

*This research was supported by the German Research Foundation through the Collaborative Research Centre SFB 401

†Institut für Gemeotrie und Praktische Mathematik, RWTH-Aachen, D-52056 Aachen, Germany, {pollul, reusken}@igpm.rwth-aachen.de

In this paper we consider the Newton-Krylov approach. A method of this class has been implemented in the QUADFLOW package, which is an adaptive multiscale finite volume solver for stationary and instationary compressible flow computations. A description of this solver is given in [2, 5, 6, 7, 30]. For the linearization we use a standard (approximate) Newton method. The resulting linear systems are solved by a preconditioned BiCGSTAB method. The main topic of this paper is a systematic comparative study of different basic preconditioning techniques. The preconditioners that we consider are based on the so-called point-block approach in which all physical unknowns corresponding to a grid point or a cell are treated as a block unknown. As preconditioners we use the point-block-Gauss-Seidel (PBGS), point-block-ILU (PBILU(0)) and point-block sparse approximate inverse (PBSPAI(0)) methods. The main motivation for considering the PBSPAI(0) preconditioner is that opposite to the other two preconditioners this method allows a trivial parallelization. We do not know of any literature in which for compressible flows the SPAI technique is compared with the more classical ILU and Gauss-Seidel preconditioners. In our comparative study we consider three test problems. The first one is a stationary Euler problem on the unit square with boundary conditions such that the problem has a trivial constant solution. For the second test problem we change the boundary conditions such that the problem has a solution consisting of three different states separated by shocks, that reflect at the upper boundary. In these first two test problems we use the Van Leer flux vector-splitting scheme [19] for discretization. We do not use an (artificial) time integration method. The third test problem is the stationary Euler flow around an NACA0012 airfoil. This standard case is also used for testing the QUADFLOW solver in [7]. Discretization is based on the flux-vector splitting method of Hänel and Schwane [13] combined with a linear reconstruction technique. For determining the stationary solution a variant of the backward Euler scheme (the b2-scheme by Batten et. al. [4]) is used. For these three test problems we perform numerical experiments for different flow conditions and varying mesh sizes. The main conclusions concerning the performance of the preconditioners are summarized at the end of section 4.

2 Discrete Euler equations

In this section we introduce three test problems that are used for a comparative study of different preconditioners. In the first two of these test problems we consider a relatively simple model situation with a standard first order flux vector-splitting scheme on a uniform grid in 2D. In the third test problem we consider more advanced finite volume techniques on locally refined grids as implemented in the QUADFLOW package.

2.1 Test problem 1: Stationary 2D Euler with constant solution

In this section we consider a very simple problem which, however, is still of interest for the investigation of properties of iterative solvers. We take $\Omega = [0,1]^2$ and consider the *stationary* Euler equations in differential form:

$$\frac{\partial f(\mathbf{u})}{\partial x} + \frac{\partial g(\mathbf{u})}{\partial y} = 0\,, \quad f(\mathbf{u}) := \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ \rho v_1 h_{tot} \end{pmatrix}, \quad g(\mathbf{u}) := \begin{pmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ \rho v_2 h_{tot} \end{pmatrix}. \qquad (1)$$

Here $\mathbf{u} = (\rho, \rho\mathbf{v}, \rho e_{tot})^T \in \Re^4$ is the vector of unknown conserved quantities: ρ denotes the density, p the pressure, $\mathbf{v} = (v_1, v_2)^T$ the velocity vector, e_{tot} the total energy and h_{tot} the total enthalpy. The system is closed by the equation of state for a perfect gas $p = \rho(\gamma - 1)(e_{tot} - 1/2|\mathbf{v}|^2)$, where γ is the ratio of specific heats, which for air has the value $\gamma = 1.4$. The boundary conditons (for the primitive variables) are taken such that these Euler equations have a constant solution. For the velocity we take $\mathbf{v} = (v_1^{in}, v_2^{in})$ with given constants $v_i^{in} > 0$, $i = 1, 2$, on the inflow boundary $\Gamma_{in} = \{ (x, y) \in \partial\Omega \mid x = 0 \text{ or } y = 0 \}$. For the density we take a constant value $\rho = \rho^{in} > 0$ on Γ_{in}. For the pressure we also take a constant value $p = \bar{p} > 0$ which is prescribed either at the inflow boundary (supersonic case) or at outflow boundary $\partial\Omega \setminus \Gamma_{in}$ (subsonic case). The Euler equations 1 then have a solution that is constant in the whole domain: $\mathbf{v} = (v_1^{in}, v_2^{in}),\ \rho = \rho^{in},\ p = \bar{p}$.

For the discretization of this problem we use a uniform mesh $\Omega_h = \{ (ih, jh) \mid 0 \leq i, j \leq n \}$, with $nh = 1$, and apply a basic upwinding method, namely the Van Leer flux vector-splitting scheme [19]. The discretization of (physical and numerical) boundary conditions is based on compatibility relations (section 19.1.2 in [14]). In each grid point we then have four discrete unknowns, corresponding to the four conserved quantities. We use a lexicographic ordering of the grid points with numbering $1, 2, \ldots, (n+1)^2 =: N$. The four unknowns at grid point i are denoted by $U_i = (u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4})^T$ and all unknowns are collected in the vector $\mathbf{U} = (U_i)_{1 \leq i \leq N}$. The discretization yields a nonlinear system of equations

$$F : \Re^{4N} \to \Re^{4N}, \quad F(\mathbf{U}) = 0 \quad . \tag{2}$$

The continuous constant solution (restricted to the grid) solves the discrete problem and thus the solution of the nonlinear discrete problem in 2 is known a-priori. This solution is denoted by $\mathbf{U}^*$. For the Jacobian $DF(\mathbf{U})$ of $F(\mathbf{U})$ explicit formulas can be derived. In section 4 we investigate the behaviour of different preconditioners when applied to a linear system of the form

$$DF(\mathbf{U}^*)\mathbf{x} = \mathbf{b} \quad . \tag{3}$$

Note that the matrix $DF(\mathbf{U}^*)$ has a regular block structure $DF(\mathbf{U}^*) = \text{blockmatrix}(A_{i,j})_{0 \leq i,j \leq N}$ with $A_{i,j} \in \Re^{4\times 4}$ for all i, j. We call this a *point-block* structure. Furthermore, $A_{i,j} \neq 0$ can occur only if $i = j$ or i and j correspond to neighbouring grid points.

2.2 Test problem 2: Stationary 2D Euler with shock reflection

We consider a two-dimensional stationary Euler problem presented in example 5.3.3 in [18]. The domain is $\Omega = [0, 4] \times [0, 1]$ and for the boundary conditions we take $\rho = 1.4$, $v_1 = 2.9$, $v_2 = 0$, $p = 1.0$ at the left boundary ($x = 0$), outflow boundary conditions at the right boundary ($x = 4$), $\rho = 2.47$, $v_1 = 2.59$, $v_2 = 0.54$, $p = 2.27$ at the lower boundary ($y = 0$) and reflecting boundary conditions at the upper boundary ($y = 1$).

With these boundary conditions the problem has a stationary solution consisting of three different states seperated by shocks, that reflect at the upper boundary, cf. figure 1. For the discretization of this problem we apply the same method as in test problem 1. This results in a nonlinear system of equations as in 2, but now the discrete solution, denoted by $\mathbf{U}^*$, is not known a-priori. In a Newton type of method applied to this nonlinear problem one has to solve linear systems with matrix $DF(\tilde{\mathbf{U}})$, $\tilde{\mathbf{U}} \approx \mathbf{U}^*$. Therefore

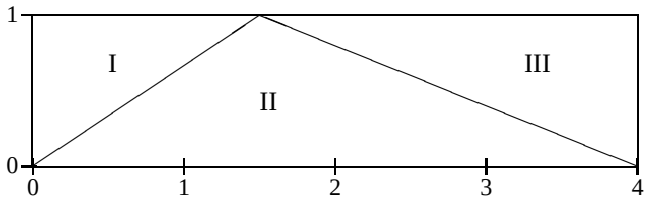

Figure 1: Three different states separated by shocks

we investigate iterative solvers applied to $DF(\mathbf{U}^*)\mathbf{x} = \mathbf{b}$. The discrete solution $\mathbf{U}^*$ is computed up to machine accuracy using some time integration method. Note that the Jacobian matrix $DF(\mathbf{U}^*)$ has a similar point-block structure as the Jacobian matrix in test problem 1.

2.3 Test problem 3: Stationary flow around NACA0012 airfoil

The third problem is a standard test case for inviscid compressible flow solvers. We consider the inviscid, transonic stationary flow around the NACA0012 airfoil (cf. [17]). The discretization is based on the conservative formulation of the Euler equations. For an arbitrary control volume $V \subset \Omega \subset \Re^2$ one has equations of the form

$$\int_V \frac{\partial \mathbf{u}}{\partial t}\, dV + \oint_{\partial V} \mathbf{F}^c(\mathbf{u})\mathbf{n}\, dS = 0 \quad . \tag{4}$$

Here $\mathbf{n}$ is the outward unit normal on ∂V, $\mathbf{u}$ the vector of unknown conserved quantities and the convective flux is given by

$$\mathbf{F}^c(\mathbf{u}) = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v} \circ \mathbf{v} + p\mathbf{I} \\ \rho h_{tot}\mathbf{v} \end{pmatrix} \quad . \tag{5}$$

The symbol $\circ$ denotes the dyadic product. As above the system is closed by the equation of state for a perfect gas. The discretization of these equations is based on cell centered finite volume schemes on an unstructured mesh as implemented in QUADFLOW and explained in [7]. For the present test problem the flux-vector splitting due to Hänel and Schwane [13] is applied. Note that this is a variant of the Van Leer flux-vector splitting method that is used in the previous two test problems. In our experiments the Van Leer method and the Hänel-Schwane method give similar results both with respect to discretization quality and with respect to the performance of iterative solvers. A linear reconstruction technique is used to obtain second order accuracy in regions where the solution is smooth. This is combined with the Venkatakrishnan limiter [34]. Although we are interested in the *stationary* solution of this problem the time derivative is not skipped. This time derivative is discretized by a time integration method which then results in a numerical method for approximating the stationary solution. To allow a fast convergence towards the stationary solution one wants to use large timesteps and thus an implicit time discretization method is preferred. This approach then results in a nonlinear system of equations in each timestep. Here we use the b2-scheme by Batten et. al. [4] for time integration. Per timestep one inexact Newton iteration is applied. In this inexact Newton method an *approximate* Jacobian is used, in which the linear reconstruction technique is neglected and the Jacobian of the first order Hänel-Schwane

discretization is approximated by one-sided difference operators (as in [33]). These Jacobian matrices have the structure

$$DF(\mathbf{U}) = \mathrm{diag}(\frac{|V_i|}{\Delta t}) + \frac{\partial R^{HS}(\mathbf{U})}{\partial \mathbf{U}} \ , \tag{6}$$

where $|V_i|$ is the volume of a control volume V_i, Δt the (local) timestep and $R^{HS}(\mathbf{U})$ the residual vector corresponding to the Hänel-Schwane fluxes. Details are given in [7]. Note that in general a smaller timestep will improve the conditioning of the approximate Jacobian in (6).

We start with an initial coarse grid and an initial CFL number C_{MIN}, which determines the size of the timestep. After each timestep in the time integration the CFL number (and thus the timestep) is increased by a constant factor until an a-priori fixed upper bound C_{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 C_{MIN}. The indicator for the local grid refinement is based on a multiscale analysis using wavelets [7]. In every timestep one approximate Newton iteration is performed. The resulting linear equation is solved with a user defined accuracy for the relative residual. If the cells are numbered $i = 1, \ldots, N$, then as in the first two test problems the approximate Jacobian has a point-block structure: $DF(\mathbf{U}) = \mathrm{blockmatrix}(A_{i,j})_{0 \le i,j \le N}$ with $A_{i,j} \in \Re^{4\times 4}$ for all i, j and $A_{i,j} \neq 0$ only if $i = j$ or i and j correspond to neighbouring cells.

3 Point-block-preconditioners

In the three test problems described above we have to solve a large sparse linear system. The matrices in these systems are sparse and have a point-block structure in which the blocks correspond to the 4 unknowns in each of the N grid points (finite differences) or N cells (finite volume). Thus we have linear systems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \ , \quad \mathbf{A} = \mathrm{blockmatrix}(A_{i,j})_{1 \le i,j \le N} \ , \quad A_{i,j} \in \Re^{4\times 4} \quad . \tag{7}$$

For the type of applications that we consider these problems are often solved by using a preconditioned Krylov-subspace method. In our numerical experiments we use the BiCGSTAB method. In the following subsections we describe basic point-block iterative methods that are used as preconditioners in the iterative solver. For the right handside we use a block representation $\mathbf{b} = (b_1, \ldots, b_N)^T$, $b_i \in \Re^4$, that corresponds to the block structure of $\mathbf{A}$. The same is done for the iterands $\mathbf{x}^k$ that approximate the solution of the linear system in 7.

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

$$P(\mathbf{A}) = \{\, (i,j) \mid A_{i,j} \neq 0 \,\} \quad . \tag{8}$$

3.1 Point-block-Gauss-Seidel method

The point-block-Gauss-Seidel method (PBGS) is the standard block Gauss-Seidel method applied to 7. 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

$$A_{i,i}x_i^{k+1} = b_i - \sum_{j=1}^{i-1} A_{i,j}x_j^{k+1} - \sum_{j=i+1}^{N} A_{i,j}x_j^k \ , \quad i = 1, \ldots, N \quad . \tag{9}$$

This method is well-defined if the 4×4 linear systems in 9 are uniquely solvable, i.e., 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 [1]. A Convergence analysis of this method for the 1D stationary Euler problem is presented in [28].

3.2 Point-block-ILU(0) method

We consider the following point-block version of the standard point ILU(0) algorithm (cf. [10, 29]):

```
for k = 1, 2, ..., N − 1
   D := A_{k,k}^{-1} ;
   for i = k + 1, k + 2, ..., N
      if (i, k) ∈ P(A)
         E := A_{i,k} D ; A_{i,k} := E ;
         for j = k + 1, k + 2, ..., N
            if (i, j) ∈ P(A) and (k, j) ∈ P(A)
               A_{i,j} := A_{i,j} − E A_{k,j};
            end if
         end j
      end if
   end i
end k
```

Figure 2: Point-block-ILU(0) algorithm

This algorithm is denoted by PBILU(0). Note that as for the PBGS method the diagonal blocks $A_{k,k}$ are assumed to be nonsingular. For this 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. As for point ILU methods one can consider variants of this algorithm in which a larger pattern as $P(\mathbf{A})$ is used and thus more fill-in is allowed (cf. for example, ILU(p), [29]). Both the PBILU(0) algorithm and such variants are available in the PETSc library.

3.3 Block sparse approximate inverse

In the SPAI method [11, 31] an approximate inverse $\mathbf{M}$ of the matrix $\mathbf{A}$ is constructed by minimizing the Frobenius norm $\|\mathbf{AM}-\mathbf{I}\|_F$ with a prescribed sparsity pattern of the matrix $\mathbf{M}$. In the point-block version of this approach (cf. [3]), denoted by PBSPAI(0), we take a block representation $\mathbf{M} = \mathrm{blockmatrix}(M_{i,j})_{1 \le i,j \le N}$, $M_{i,j} \in \Re^{4\times 4}$, and the set of admissible approximate inverses is given by

$$\mathcal{M} := \{\, \mathbf{M} \in \Re^{4N \times 4N} \mid P(\mathbf{M}) \subseteq P(\mathbf{A}) \,\} \quad . \tag{10}$$

A sparse approximate inverse $\mathbf{M}$ is determined by minimization over this admissible set

$$\|\mathbf{AM} - \mathbf{I}\|_F = \min_{\tilde{\mathbf{M}} \in \mathcal{M}} \|\mathbf{A}\tilde{\mathbf{M}} - \mathbf{I}\|_F \quad .$$

The choice for the Frobenius norm allows a splitting of this minimization problem. Let $\tilde{\mathbf{M}}_j = \mathrm{blockmatrix}(\tilde{M}_{i,j})_{1 \le i \le N} \in \Re^{4N \times 4}$ be the j-th block column of the matrix $\tilde{\mathbf{M}}$ and $\mathbf{I}_j$ the corresponding block column of $\mathbf{I}$. Let $\tilde{m}_{j,k}$ and $e_{j,k}$, $k = 1, \ldots, 4$, be the k-th columns of the matrix $\tilde{\mathbf{M}}_j$ and $\mathbf{I}_j$, respectively. Due to

$$\|\mathbf{A}\tilde{\mathbf{M}} - \mathbf{I}\|_F^2 = \sum_{j=1}^{N} \|\mathbf{A}\tilde{\mathbf{M}}_j - \mathbf{I}_j\|_F^2 \quad = \sum_{j=1}^{N} \sum_{k=1}^{4} \|\mathbf{A}\tilde{m}_{j,k} - e_{j,k}\|_2^2 \tag{11}$$

the minimization problem can be split into $4N$ decoupled least squares problems:

$$\min_{\tilde{m}_{j,k}} \|\mathbf{A}\tilde{m}_{j,k} - e_{j,k}\|_2 \,, \quad j = 1, \ldots, N, \quad k = 1, \ldots, 4. \tag{12}$$

The vector $\tilde{m}_{j,k}$ has the block representation $\tilde{m}_{j,k} = (m_1, \ldots, m_N)^T$ with $m_\ell \in \Re^4$ and $m_\ell = 0$ if $(\ell, j) \notin P(\mathbf{A})$. Hence for fixed (j,k) and with $e_{j,k} =: (e_1, \ldots, e_N)^T$, $e_\ell \in \Re^4$, we have

$$\|\mathbf{A}\tilde{m}_{j,k} - e_{j,k}\|_2^2 = \sum_{i,\ell=1}^{N}{}^{*} \|A_{i,\ell} m_\ell - e_\ell\|_2^2$$

where in the double sum $\sum^{*}$ only pairs (i,ℓ) occur with $(i,\ell) \in P(\mathbf{A})$ and $(\ell, j) \in P(\mathbf{A})$. Thus the minimization problem for the column $\tilde{m}_{j,k}$ in (12) is a *low dimensional* least squares problem that can be solved by standard methods. Due to (11) these least squares problems for the different columns of the matrix $\mathbf{M}$ *can be solved in parallel*. Moreover the application of the PBSPAI(0) preconditioner requires a sparse matrix-vector product computation which is 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.

In the literature a row-variant of SPAI is also used. This method is based on the minimization problem

$$\|\mathbf{MA} - \mathbf{I}\|_F = \min_{\tilde{\mathbf{M}} \in \mathcal{M}} \|\tilde{\mathbf{M}}\mathbf{A} - \mathbf{I}\|_F$$

A row-wise decoupling leads to a very similar method as the one described above. Here we denote this algorithm by PBSPAI$_{row}$(0).
As for the ILU preconditioner these SPAI preconditioners have variants in which additional fill-in is allowed, cf. [3, 11]. Besides as a preconditioner the SPAI method can also be used as a smoother in multigrid solvers (cf. [9, 32]).

4 Numerical experiments

In this section we present results of numerical experiments. Our goal is to illustrate and to compare the behaviour of the different preconditioners presented above for a few test

problems. The first two test problems (described in section 2.1 and 2.2) are Jacobian systems that result from the Van Leer flux vector-splitting discretization on a uniform mesh with mesh size h, cf. (3). These Jacobians are evaluated at the discrete solution $\mathbf{U}^*$. In the first test problem this solution is a trivial one (namely, constant), whereas in the second test problem we have a reflecting shock. In the third test case we have linear systems with matrices as in (6) that arise in the solver used in the QUADFLOW package.

In all experiments below we use a left preconditioned BiCGSTAB method. For the first two test problems, the discretization routines, methods for the construction of the Jacobian matrices and the preconditioners (PBGS, PBILU(0) and PBSPAI(0)) are implemented in MATLAB. We use the BiCGSTAB method available in MATLAB. For the third test problem the approximate Jacobian matrices as in (6) are computed in QUADFLOW. For the preconditioned BiCGSTAB method and the PBGS, PBILU(p), $p = 0, 1, 2$, preconditioners we use routines from the PETSc library [1].

To measure the quality of the preconditioners we present the number of iterations that is needed to satisfy a certain tolerance criterion. To allow a fair comparison of the different preconditioners 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 $\mathbf{A}$ (=: 1 matvec). For the PBGS method we have no construction costs. The arithmetic work per application of PBGS is about 0.7 matvec. Note that both in PBILU(0) and PBSPAI(0) the nonzero *block*-pattern is not larger that of the matrix $\mathbf{A}$ (e.g., for PBSPAI(0), $P(\mathbf{M}) \subseteq P(\mathbf{A})$ in (10)). However, in a nonzero block of the matrix $\mathbf{A}$ certain entries can be zero, whereas in the preconditioner the corresponding entries may be nonzero. For example, in the shock reflection test problem, about one fourth of the entries in the nonzero blocks are zero, whereas in the PBILU(0) and PBSPAI(0) preconditioners for this problem almost all entries in the nonzero blocks are nonzero. 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 PBILU(0). The costs for constructing the PBSPAI(0) preconditioner are much higher (note, however, the high parallalization potential). Typical values (depending on $P(\mathbf{A})$) in our experiments are 20-50 matvecs. In the application of this preconditioner no 4×4 subproblems have to be solved and due to this the arithmetic work is somewhat less as for the PBILU(0) preconditioner. We typically need 1.2-1.5 matvecs per application of the PBSPAI(0) preconditioner. Summarizing, if we only consider the costs per application of the preconditioners than the PBILU(0) method is about twice as expensive as the PBGS method and the PBSPAI(0) preconditioner is slightly less expensive than the PBILU(0) method.

4.1 Test problem 1: Stationary 2D Euler with constant solution

We consider the discretized stationary Euler equations described in test problem 1 with mesh size $h = 0.02$. We vary the Mach number in x-direction, which is denoted by M_x: $0.05 \leq M_x \leq 1.25$. For the Mach number in y-direction, denoted by M_y, we take $M_y = \frac{3}{2}M_x$. The linear system in (3) is solved with the preconditioned BiCGSTAB method, with starting vector zero. The iteration is stopped if the relative residual (2-norm) is below 1E-6. Results are presented in Figure 3. 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 shows an undesirable very irregular behaviour. There are peaks in the iteration counts close to $M_x = \frac{2}{3}$ (hence, $M_y = 1$) and $M_x = 1$. Applying BiCGSTAB without preconditioning we observe divergence for most M_x values and if the method converges then its rate of convergence is extremely low.

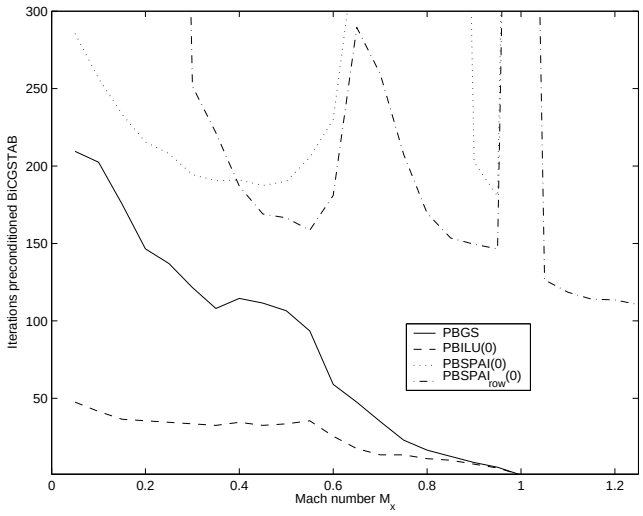

Figure 3: Test problem 1, iteration count

4.2 Test problem 2: Stationary 2D Euler with shock reflection

We consider the discretized stationary Euler equations with a shock reflection as described in section 2.2 on grids with different mesh sizes $h_x = h_y = 2^{-k},\ k = 3, \ldots, 7$. The discrete solution $\mathbf{U}^*$ is determined with high accuracy using a damped Newton method. As in section 4.1 we use the preconditioned BiCGSTAB method to solve a linear system with matrix $DF(\mathbf{U}^*)$ until the relative residual is below 1E-6. The number of iterations that is needed is shown in table 1. The symbol † denotes that the method did not converge within 2000 iterations. Both for the PBGS and the PBILU(0) preconditioner we observe the expected h_y^{-1} behavior in the iteration counts. With PBGS preconditioning one needs about 1.5 to 2 times as much iterations as with PBILU(0) preconditioning. Again the performance of the PBSPAI(0) preconditioner is very poor.

4.3 Test Problem 3: Stationary flow around NACA0012 airfoil

We consider two standard NACA0012 airfoil test cases ([17] testcases 3 and 1; the other three reference test cases for this airfoil yield similar results) as described in test

mesh size h_y	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
PBGS	15	26	51	108	232
PBILU(0)	10	17	30	59	109
PBSPAI(0)	183	†	†	†	†
PBSPAI $_{row}$(0)	193	†	†	†	†

Table 1: Test problem 2, iteration count

problem 3. Test case A corresponds to the flow parameters $M_\infty = 0.95,\ \alpha = 0^\circ$ (also used in [7]) and test case B corresponds to the parameters $M_\infty = 0.8,\ \alpha = 1.25^\circ$. Related to the discretization we recall some facts from [7]. The far-field boundary is located about 20 chord lenghts from the airfoil. Standard characteristic boundary conditions are applied at the far-field. Computations are initialized on a structured grid consisting of 4 blocks with 10×10 cells each. Adaptation of the grid was performed each time the density residual has decreased two orders of magnitude. On the finest level the iteration is stopped when the density residual is reduced by a factor of 10^4. The grid refinement and coarsening is controlled by a multiscale wavelet technique. For test case A this results in a sequence of 14 (locally refined) grids. In table 2 we show the number of cells in each of these grids. We note that on the finer grids a change

Grid	1	2	3	4	5	6	7
# cells	400	1,600	4,264	7,006	11,827	15,634	21,841
Grid	8	9	10	11	12	13	14
# cells	25,870	28,627	30,547	31,828	33,067	33,955	34,552

Table 2: Test case A, sequence of grids

(refinement/coarsening) to the next grid does not necessarily imply a smaller finest mesh size. It may happen that only certain coarse cells are refined to obtain a better shock resolution. For a discussion of this adaptivity issue we refer to [7]. In figure 4 the final grid (34,552 cells) and the corresponding Mach number distribution is shown. The flow pattern downstream of the trailing edge has a complex shock configuration. Two oblique shocks are formed at the trailing edge. The supersonic region behind these oblique shocks is closed by a further normal shock.

On each grid an implicit time integration is applied (cf. section 2.3). Currently the choice of the timestep is based on an ad hoc strategy. Starting with a CFL number equal to 1 the timestep is increased based on the rule $\mathrm{CFL}_{\mathrm{new}} = 1.1\,\mathrm{CFL}_{\mathrm{old}}$. A maximum value $\mathrm{CFL} = 1000$ is allowed. Per timestep one inexact Newton iteration is applied. The linear systems with the approximate Jacobians (cf. section 2.3) are solved by a preconditioned BiCGSTAB method until the relative residual is smaller than 1E-2. Because the PBSPAI(0) preconditioners have shown a very poor behaviour already for the relatively simple problems in section 4.1 and section 4.2 we decided not to use the PBSPAI(0) preconditioner in this test problem. An interface between QUADFLOW and the PETSc library [1] has been implemented. This makes the BiCGSTAB method and PBILU(0) preconditioner available. Recently also a PBGS routine has become available. The PETSc library offers many other iterative solvers and preconditioners. Here, besides the PBGS and PBILU(0) preconditioners we also consider block variants of ILU that allow more fill-in, namely the PBILU(1) and PBILU(2) methods.

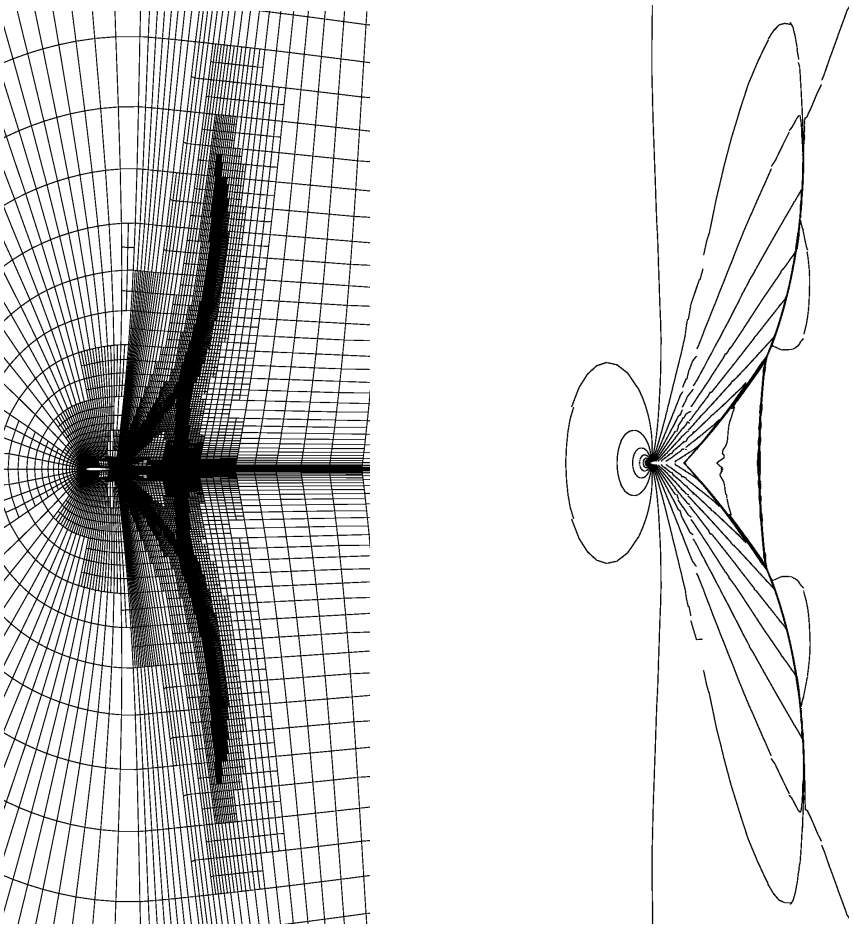

Figure 4: Test case A. Left figure: computational grid. Right figure: Mach distribution, $M_{min} = 0.0$, $M_{max} = 1.45$, $\Delta M = 0.05$

The arithmetic work in the QUADFLOW solver is dominated by the linear solves on the finest grids. We present results only for the two finest grids. In figure 5, for each timestep on these two grids the corresponding number of preconditioned BiCGSTAB iterations is given. Note that different preconditioners may leed to (slightly) different numbers of timesteps. On grid 13 we have about 110 timesteps and then a change to grid 14 takes place. In these 110 timesteps on grid 13 the iteration count shows a clear increasing trend. This is due to the increase of the CFL number. After the change to grid 14 one starts with $\mathrm{CFL} = 1$ and a similar behaviour occurs again.

In test case B the adaptivity strategy results in 11 grids. In table 3 for both test cases we show the *averaged* number of preconditioned BiCGSTAB iterations for the two finest grids, where the average is taken over the time steps per grid.

Note that with PBGS we need 2-2.5 times more iterations as with PBILU(0) and with PBILU(2) we need about 2 times less iterations as with PBILU(0). Taking the arithmetic work per iteration into account we conclude that PBGS and PBILU(0) have comparable efficiency, whereas the PBILU(p), $p = 1, 2$, preconditioners are (much) less efficient.

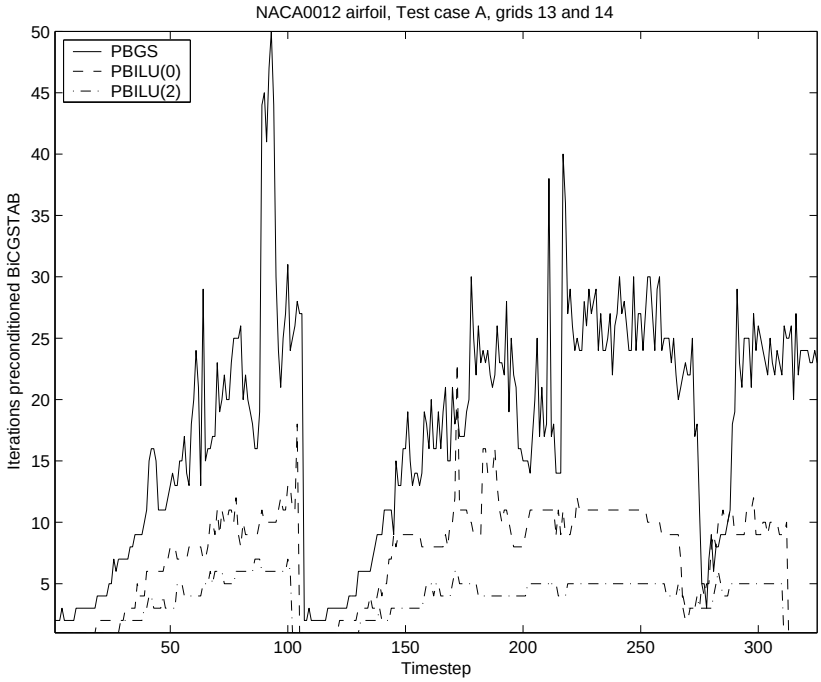

Figure 5: Test case A, iteration count

test case	A		B	
Grid	13	14	10	11
PBGS	14.9	18.6	2.89	27.0
PBILU(0)	6.17	8.37	1.33	9.83
PBILU(1)	4.24	4.65	1.04	6.07
PBILU(2)	3.52	3.83	1.00	5.09

Table 3: Test cases A and B, avarage iteration count

To illustrate the dependence of the rate of convergence on the mesh size we consider, for test case A, a sequence of uniformly refined grids, cf. table 4. On every grid

Grid	1	2	3	4	5
# cells	400	1,600	6,400	25,600	102,400

Table 4: sequence of uniformely refined grids

we continue the time integration until the density residual has decreased two orders of magnitude. Iteration counts are shown in table 5.

From these results we see that due to the mass matrix coming from the (artificial) time integration the average iteration count increases much slower as h^{-1}. The total number of iterations, however, shows a clear h^{-1} behaviour as in the model problem in section 4.2. The large total number of iterations needed on fine grids (table 5, right) is caused by the many timesteps that are needed. A significant improvement of efficiency may come from a better strategy for the timestep control.

Grid	1	2	3	4	5
PBGS	5.8	9.2	13.3	18.2	21.7
PBILU(0)	1.9	2.4	2.9	4.3	4.5
PBILU(1)	1.6	1.7	1.9	2.7	2.9
PBILU(2)	1.0	1.4	1.8	1.9	2.0

Grid	1	2	3	4	5
PBGS	947	1806	3739	9030	19212
PBILU(0)	301	462	806	2121	4025
PBILU(1)	250	335	547	1324	2562
PBILU(2)	161	274	510	940	1783

Table 5: Test case A. Average iteration count (above) and sum over all time steps per grid (below).

Summary

We summarize the main conclusions. Already for our relatively simple model problems the PBSPAI(0) method turns 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 for NACA0012 airfoil) 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. If one applies an artificial time integration method for solving a stationary compressible flow problem then the timestep control strategy strongly influences the overall efficiency of the solver.

Acknowledgement

The experiments in section 4.3 are done using the QUADFLOW solver developed in the Collaborative Research Center SFB 540. The authors acknowledge the fruitful collaboration with several members of the QUADFLOW research group.

References

[1] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes. B. F. Smith and H. Zhang, PETSc, http://www-fp.mcs.anl.gov/petsc/ (2001).

[2] J. Ballmann (editor), Flow Modulation and Fluid-Structure-Interaction at Airplane Wings, Numerical Notes on Fluid Mechanics **84**, Springer (2003).

[3] S. T. Barnard and M. J. Grote, A Block Version of the SPAI Preconditioner, in Proc. 9th SIAM Conf. On Parall. Process. for Sci. Comp. (1999).

[4] P. Batten, M. A. Leschziner and U.C. Goldberg, Average-State Jacobians and Implicit Methods for Compressible Viscours and Turbulent Flows, J. Comp. Phys., **137**, 38–78 (1997).

[5] K.H. Brakhage and S. Müller, Algebraic-hyperbolic Grid Generation with Precise Control of Intersection of Angles, Int. J. Numer. Meth. Fluids., **33**, 89–123 (2000).

[6] F. Bramkamp, B. Gottschlich-Müller, M. Hesse, Ph. Lamby, S. Müller, J. Ballmann, K.-H. Brakhage, W. Dahmen, H-adaptive Multiscale Schemes for the Compressible Navier-Stokes Equations: Polyhedral Discretization, Data Compression and Mesh Generation, in [2], 125–204 (2001).

[7] F. Bramkamp, Ph. Lamby and S. Mller, An adaptive multiscale finite volume solver for unsteady and steady state flow computations, Journal of Computational Physics, **197/2** 460–490 (2004).

[8] A. Brandt, Multi-level Adaptive Solutions to Boundary Value Problems, Math. Comp., **31**, 333–390 (1977).

[9] O. Bröker, M.J. Grote, C. Mayer, A. Reusken, Robust Parallel Smoothing for Multigrid via Sparse Approximate Inverses, SIAM J. Sci. Comput., **23** 1396–1417 (2001).

[10] J. J. Dongarra, I. S. Duff, D. C. Sorensen and H. A. van der Vorst, Numerical Linear Algebra for High-Performance Computers, Software-Environments-Tools, SIAM, Philadelphia, PA (1998).

[11] M.J. Grote and T. Huckle, Parallel Preconditioning with Sparse Approximate Inverses, SIAM J. of Scient. Comput., **18(3)** (1997).

[12] W. Hackbusch, Multi-grid Methods and Applications, Springer (1985).

[13] D. Hänel and F. Schwane, An Implicit Flux-Vector Splitting Scheme for the Computation of Viscous Hypersonic Flow, AIAA paper 0274 (1989).

[14] C. Hirsch, Numerical computation of internal and external flows: computational methods for inviscid and viscous flows, Vol. 2., Wiley (1988).

[15] A. Jameson, Solution of the Euler Equations for Two-Dimensional Transonic Flow by a Multigrid Method, Appl. Math. Comp. **13**, 327–356 (1983).

[16] A. Jameson, D.A. Caughey, How Many Steps are Required to Solve the Euler Equations of Steady, Compressible Flow: in search of a Fast Solution Algorithm, AIAA paper 2673 (2001).

[17] D. J. Jones, Reference Test Cases and Contributors, Test Cases For Inviscid Flow Field Methods. AGARD Advisory Report **211**(5) (1986).

[18] D. Kröner, Numerical schemes for conservation laws, Wiley (1997).

[19] B. van Leer, Flux Vector Splitting for the Euler Equations. In: Proceedings of the 8th International Conference on Numerical Methods in Fluid Dynamics (E. Krause, ed.). Lecture Notes in Physics, **170** 507–512. Springer, Berlin (1982).

[20] B. van Leer and D. Darmofal, Steady Euler Solutions in $O(N)$ Operations, Multigrid Methods (E. Dick, K. Riemslagh and J. Vierendeels, editors) **VI** 24–33 (1999).

[21] I. Lepot, P. Geuzaine, F. Meers, J.-A. Essers and J.-M. Vaassen, Analysis of Several Multigrid Implicit Algorithms for the Solution of the Euler Equations on Unstructured Meshes, Multigrid Methods (E. Dick, K. Riemslagh and J. Vierendeels, editors) **VI** 157–163 (1999).

[22] H. Luo, J. Baum and R. Loehner, A Fast, Matrix-free Implicit Method for Compressible Flows on Unstructured Grids, J. of Comp. Physics, **146** 664-690 (1998).

[23] D.J. Mavripilis and V. Venkatakrishnan, Implicit method for the Computation of Unsteady Flows on Unstructured Grids, J. Comp. Phys., **127** 380–397 (1996).

[24] P.R. McHugh and D.A. Knoll, Comparison of Standard and Matrix-Free Implementations of Several Newton-Krylov Solvers, AIAA J., **32** 394–400 (1994).

[25] A. Meister, Comparison of different Krylov Subspace Methods Embedded in an Implicit Finite Volume Scheme for the Computation of Viscous and Inviscid Flow Fields on Unstructured Grids, J. of Comp. Phys., **140** 311–345 (1998).

[26] A. Meister, Th. Sonar, Finite-Volume Schemes for Compressible Fluid Flow. Surv. Math. Ind., **8** 1–36 (1998).

[27] A. Meister, C. Vömel, Efficient Preconditioning of Linear Systems Arising from the Discretization of Hyperbolic Conservation Laws, Advances in Computational Mathematics, **14** 49–73 (2001).

[28] A. Reusken, Convergence analysis of the Gauss–Seidel Preconditioner for Discretized One Dimensional Euler Equations, SIAM J. Numer. Anal., **41** 1388–1405 (2003).

[29] Y. Saad, Iterative methods for sparse linear systems, PWS Publishing Company, Boston (1996).

[30] SFB 401, Collaborative Research Center, Modulation of flow and fluid-structure interaction at airplane wings, RWTH Aachen University of Technology, http://www.lufmech.rwth-aachen.de/sfb401/kufa-e.html

[31] W.-P. Tang, Toward an Effective Approximate Inverse Preconditioner, SIAM J. Matrix Anal. Appl., **20** 970–986, (1999).

[32] W.-P. Tang and W. L. Wan, Sparse Approximate Inverse Smoother for Multigrid, SIAM J. Matrix Anal. Appl., **21** 1236–1252 (2000).

[33] K.J. Vanden and P. D. Orkwis, Comparison of Numerical and Analytical Jacobians, AIAA Journal, **34(6)** 1125–1129 (1996).

[34] V. Venkatakrishnan, V., Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters, J. Comp. Phys., **118** 120–130 (1995).

[35] V. Venkatakrishnan, Implicit schemes and Parallel Computing in Unstructured Grid CFD, ICASE-report 28 (1995).