Preconditioner Updates Applied to CFD Model Problems

Philipp Birken, Jurjen Duintjer Tebbens, Andreas Meister and Miroslav Tůma

January 31, 2007

The work of the first and third author is supported by the German Science Foundation as part of the Sonderforschungsbereich SFB/TR TRR 30 „Prozessintegrierte Herstellung funktional gradierter Strukturen auf der Grundlage thermo-mechanisch gekoppelter Phänomene", project C2. The work of the second and fourth author is supported by the Program Information Society under project 1ET400300415. The work of the second author is also supported by project number KJB100300703 of the Grant Agency of the Academy of Sciences of the Czech Republic.

Abstract

In the present paper we concentrate on solving sequences of nonsymmetric linear systems with block structure arising from compressible flow problems. We attempt to improve the solution process by sharing part of the computational effort throughout the sequence. This is achieved by application of a cheap updating technique for preconditioners which we adapted in order to be used for our applications. Tested on three benchmark compressible flow problems, the strategy speeds up the entire computation with an acceleration being particularly pronounced in phases of instationary behavior.

Keywords: Finite volume methods; Update preconditioning; Krylov subspace methods; Euler equations; Conversation laws

1 Introduction

Standard discretization schemes for both stationary and instationary problems in aerodynamics are finite volume methods. As the CFL condition puts a severe restriction on the time step of explicit methods, time integration is often done implicitly. Using Newton's method for the appearing nonlinear equation systems, the problem of solving a partial differential equation numerically is transformed into the problem of solving a sequence of linear equation systems. In general, up to 80% of the CPU time for a flow solver is spent solving the linear systems. Thus, the major bottleneck in numerical simulation is the solution of the sequence of linear systems and there is a continuous demand to improve upon the existing methods.

Popular methods used in solving the large and sparse linear systems involved here include multigrid methods and Krylov subspace methods. Multigrid methods use multiple discretization levels and combine several techniques on the different levels (see, e.g. [23]). For some important classes of problems they are asymptotically optimal, but they can also be sensitive to changes of the problem [8]. Krylov subspace methods are based on projecting the large linear system to subspaces of small dimension (see, e.g. [21]). The subspaces are generated through multiplication of vectors by the system matrix, thus enabling exploitation of sparsity. In favorable cases, dominant properties become apparent at an early stage of computation and a satisfactory approximation to the solution can be obtained in a relatively small number of iterations. In practice, one often combines multigrid methods with Krylov subspace methods by using a method of one class as a preconditioner for a method of the other class (see, e.g. [26]). We will consider here Krylov subspace methods, but the techniques we describe may also be applied to other solvers. For the non-normal linear systems that we have to solve, basically two classes of Krylov subspace methods may be used. In the first class, whose main representant is the GMRES method [22], we find methods that reduce residual norms in every iteration, but that must be restarted for reasons of storage and computational costs. The second class contains methods like BiCGSTAB [24], working with short recurrences but without guarantee that the process does not start to oscillate or does not break down. Often more important than the choice of the specific Krylov subspace method used is the choice of the preconditioner for the linear systems. For our problems,

incomplete factorizations lead to good results that are in many cases hard to improve.

In order to speed up the solution process of the linear systems arising in CFD problems, we will not search for new and even more sophisticated linear solvers or preconditioners in this paper. Instead, we will try to accelerate the existing methods by considering the whole sequence of linear systems and by trying to share some of the computational effort throughout the sequence. In stationary and instationary problems linear systems are often close during many subsequent iterations of the nonlinear process. A well-known way to exploit this is by skipping some evaluations of the Jacobian in Newton's method, changing only the right hand sides. Unfortunately, this leads to weaker convergence of the nonlinear process. Concerning preconditioning, closeness of system matrices has been taken advantage of only in a rather naive way. Very often, a preconditioner is recomputed periodically with some heuristic choice of period, and at a certain point it may be completely frozen [16].

In recent years, a few attempts to *update* preconditioners for large sparse systems have been made in the numerical linear algebra community. The main idea is to derive efficient preconditioners from previous systems of the sequence in a cheap way, thus avoiding the expensive computation of a new preconditioner. For instance, in case of a sequence of linear systems from a quasi-Newton method, straightforward approximate small rank updates can be useful (this is shown in the SPD case in [17], [5]). In [3, 6] approximate diagonal preconditioner updates were introduced for sequences of parametric complex symmetric linear systems. This technique was generalized to approximate (possibly permuted) triangular updates for nonsymmetric sequences in [7]. Finally, recycling of Krylov subspaces by using adaptive information generated during previous runs has been used to update both preconditioners and Krylov subspace iterations (see [19], [12], [18] and [2]). Note that from the mentioned techniques only the last two are designed for sequences of nonsymmetric linear systems.

In this paper we investigate the effect of updating preconditioners on the speed of the solution process for some model problems from CFD. These are chosen from a broad range of Mach numbers to represent different wellknown types of problems. The model problems lead to nonsymmetric linear systems and we will update the corresponding preconditioners based on the technique proposed in [7]. To our knowledge, this kind of strategy is applied to the CFD model problems for the first time. We will describe how we adapted the original technique in order to be used for the model problems. Then we demonstrate that the technique is able to speed up the solution of the involved linear systems, with an acceleration being particularly significant in phases with important changes between subsequent system matrices. In the next section we address the governing equations and the discretization we used for the numerical solution process. In Section 3 we say some words about solving the linear systems in general and then concentrate on the update technique. Among others, we present some theoretical results and a detailed overview of the modifications for block systems. In Section 4 we display and discuss the results of numerical experiments with the model problems. Unless otherwise stated, $\|\cdot\|$ denotes an arbitrary matrix norm.

2 Governing Equations and Finite Volume Discretization

2.1 The Euler Equations

The equations governing our model problems are the Euler equations. These consist of the conservation laws of mass, momentum and energy, closed by an equation of state. We consider only the two dimensional case. Given an open domain $D \subset \mathbb{R}^2$, the equations can be expressed as

$$\partial_t \mathbf{u} + \sum_{j=1}^{2} \partial_{x_j} \mathbf{f}_j(\mathbf{u}) = \mathbf{0} \quad \text{in } D \times \mathbb{R}^+,$$

where $\mathbf{u} = (\rho, m_1, m_2, \rho E)^T$ represents the vector of conserved variables. The flux functions $\mathbf{f}_j$ are given by

$$\mathbf{f}_j(\mathbf{u}) = \begin{pmatrix} m_j \\ m_j v_1 + \delta_{1j} p \\ m_j v_2 + \delta_{2j} p \\ H m_j \end{pmatrix}, \; j = 1, 2,$$

with δ_{ij} denoting the Kronecker symbol. The quantities ρ, $\mathbf{v} = (v_1, v_2)^T$, $\mathbf{m} = (m_1, m_2)^T$, E and $H = E + \frac{p}{\rho}$ describe the density, velocity, momentum per unit volume, total energy per unit mass and total enthalpy per unit mass, respectively. The pressure is defined by the equation of state for a perfect gas $p = (\gamma - 1)\rho(E - \frac{1}{2}|\mathbf{v}|^2)$, where γ denotes the ratio of specific heats, taken as 1.4 for air.

2.2 The Finite Volume Method

We will use here a finite volume discretization. Finite volume schemes are based on the integral form of the governing equations. As this approach is covered extensively in the literature [11][10],[15] we will give only a short summary of the specific concepts used.

Our spatial discretization of the physical domain into control volumes or cells σ_i is constructed as a secondary mesh from an underlying Delaunay-triangularization, see figure 1 (left).

We call $\mathbf{u}$ a weak solution of the Euler equations, if

$$\frac{d}{dt} \int_\sigma \mathbf{u} \, dx + \oint_{\partial\sigma} \{\mathbf{f}_1(\mathbf{u}) n_1 + \mathbf{f}_2(\mathbf{u}) n_2\} \, ds = 0 \tag{1}$$

holds for all control volumes σ with outer unit normal vector $\mathbf{n} = (n_1, n_2)^T$.

For a control volume σ_i with volume $|\sigma_i|$, let $N(i)$ denote the set of its neighbors. Note that the boundary between two neighbors σ_i and σ_j consists usually of two line segments l_{ij}^1 and l_{ij}^2 according to figure 1 (right).

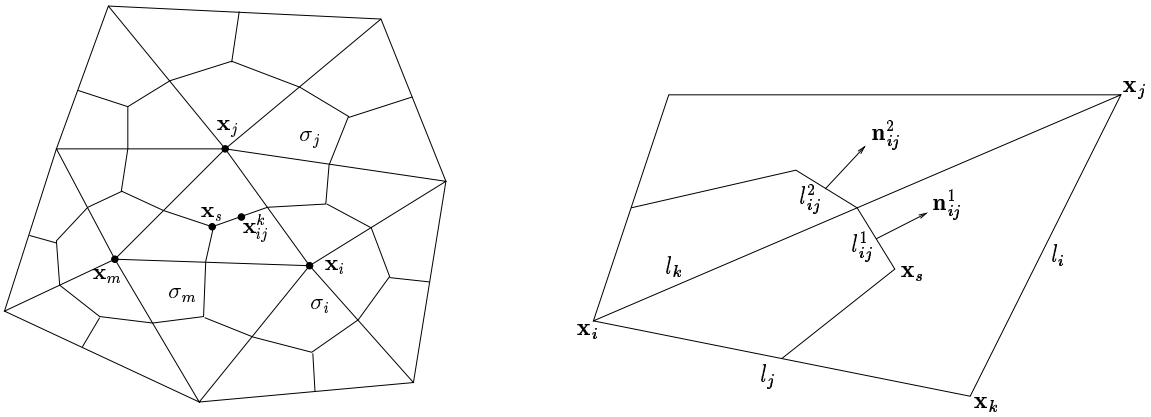


Figure 1: Triangularization and boxes (left). Geometry between boxes (right)

Reformulating (1) for σ_i under the assumption that the mesh does not change with time results in

$$\frac{d}{dt}\mathbf{u}_i(t) = -\frac{1}{|\sigma_i|}\sum_{j\in N(i)}\sum_{k=1}^{2}\int_{l_{ij}^k}\sum_{\ell=1}^{2}\mathbf{f}_\ell(\mathbf{u})\mathbf{n}_{ij,\ell}^k\,ds. \tag{2}$$

We now consider the mean value $\mathbf{u}_i(t) := \frac{1}{|\sigma_i|}\int_{\sigma_i}\mathbf{u}\,dx$ in each cell and obtain an evolution equation for the cell averages on σ_i. Using a second order Gaussian quadrature rule with Gauss point x_{ij}^k for the line integrals yields

$$\frac{d}{dt}\mathbf{u}_i(t) = -\frac{1}{|\sigma_i|}\sum_{j\in N(i)}\sum_{k=1}^{2}\frac{|l_{ij}^k|}{2}\left(\sum_{\ell=1}^{2}\mathbf{f}_\ell\left(\mathbf{u}(x_{ij}^k,t)\right)\mathbf{n}_{ij,\ell}^k\right).$$

The pointwise evaluation of the fluxes is replaced by a numerical flux function $\mathbf{H}$, which we have chosen to be AUSMDV from [25] or for low Mach numbers a Lax-Friedrichs-type flux developed for these cases [14]. Then, the cell averages are introduced as arguments and we obtain the following evolution equation:

$$\frac{d}{dt}\mathbf{u}_i(t) = -\frac{1}{|\sigma_i|}\sum_{j\in N(i)}\sum_{k=1}^{2}|l_{ij}^k|\mathbf{H}(\mathbf{u}_i(t),\mathbf{u}_j(t);\mathbf{n}_{ij}^k). \tag{3}$$

Computing the solution with a piecewise constant approximation to $\mathbf{u}(\mathbf{x},t)$ results in a method that can be at most of first order. To obtain higher order, we will use a reconstruction technique which uses a linear representation $\mathbf{u}_i(t)$ of $\mathbf{u}(\mathbf{x},t)$ in each cell. As this leads to spurious oscillations near shocks, the Barth-Jespersen-limiter is used to reduce the spatial discretization to first order where necessary.

Implicit time stepping schemes inherently fulfill the CFL-condition, since the numerical domain of dependence always covers the physical one. The application of such a scheme leads to a nonlinear system of equations. As we consider only steady state problems, we employ the implicit Euler scheme and obtain:

$$\mathbf{\Omega}\,\mathbf{u}^{n+1} = \mathbf{\Omega}\,\mathbf{u}^{n} + \Delta t \mathbf{H}(\mathbf{u}^{n+1}),$$

where $\mathbf{u}$ is the vector of the conservative variables from all cells. Correspondingly, $\mathbf{f}(\mathbf{u})$ denotes an evaluation of the numerical flux function on the whole grid. $\mathbf{\Omega}$ is the diagonal matrix of the volumes of the cells, corresponding to the variables in $\mathbf{u}$. This equation is solved approximately using one step of Newton's method, which is sufficient for steady state problems. For unsteady problems more steps are often required and the extension of the method is straightforward. The starting value here is $\mathbf{u}^n$ and the corresponding linear system of equations can be written as (see (3))

$$\mathbf{A}\Delta\mathbf{u} = \mathbf{rhs}(\mathbf{u}^n), \text{ where } \mathbf{A} = \left[\mathbf{\Omega} + \Delta t \frac{\partial \mathbf{H}(\mathbf{u})}{\partial \mathbf{u}}\right]_{\mathbf{u}^n}, \tag{4}$$

with the update $\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta\mathbf{u}$. The matrix $\mathbf{A} = (\mathbf{A}_{ij})$ has a block structure, where each element $\mathbf{A}_{ij} \in \mathbb{R}^{4\times 4}$ vanishes if the corresponding control volumes σ_i and σ_j are not adjacent. Clearly, $\mathbf{A}$ represents a large and sparse matrix. As the involved grid is in general unstructured, so is the sparsity pattern of $\mathbf{A}$. Note that the sparsity pattern of these matrices remains the same during all time steps. Whereas in some cases at least the pattern is symmetric, usually the matrix itself is nonsymmetric. From (4) we can deduce that the matrix is close to a block diagonal matrix for small time steps and small derivatives of $\mathbf{H}(\mathbf{u})$. Diagonal dominance implies some attractive properties of preconditioners and iterative solvers; however, in our model problems the dominance is too weak to take advantage of.

3 Iterative Solution of the Involved Systems

3.1 Preconditioned Krylov Subspace Methods

As we mentioned in the introduction, we will solve the linear systems from (4) with Krylov subspace methods. For ease of notation, we here denote linear systems from (4) with $\mathbf{Ax} = \mathbf{b}$ and assume their dimension is N (not to be confounded with the set of neighbors $N(i)$ from Section 2). In theory, all Krylov subspace methods work with the same subspaces of the form $\mathcal{K}_k(\mathbf{A}, \mathbf{r_0}) = \text{span}\{\mathbf{r_0}, \mathbf{Ar_0}, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\}$, $k < N$, where $\mathbf{r_0} = \mathbf{b} - \mathbf{Ax_0}$ is the initial residual for an initial guess $\mathbf{x_0} \in \mathbb{R}^N$. The initial residual is projected onto subsequent spaces of the form $\mathbf{A}\mathcal{K}_k(\mathbf{A}, \mathbf{r_0})$ and the iterative process is stopped when the difference between $\mathbf{r_0}$ and its projection is smaller than a prescribed tolerance. Individual methods differ in the kind of projection that is used, in the type of basis for $\mathcal{K}_k(\mathbf{A}, \mathbf{r_0})$ that is generated and in implementational aspects. For an overview of Krylov subspace methods see for example [21]. For the nonsymmetric matrices we have here, the choices of Krylov subspace methods are somewhat limited. A popular and efficient method with low demands on storage costs is the BiCGSTAB method [24]. Whereas the similarly popular GMRES method [22] has some other advantages that we explained in the introduction, we concentrate here on BiCGSTAB because for our finite volume scheme it has turned out to be slightly faster than GMRES.

Of major importance for the performance of Krylov subspace methods is the right choice

of the preconditioner. Preconditioners transform the linear system in an attempt to create better convergence properties when the iterative method is applied. In its most general form, the preconditioned linear system can be written as

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\mathbf{x}^P = \mathbf{M}_L^{-1}\mathbf{b}, \quad \mathbf{x} = \mathbf{M}_R^{-1}\mathbf{x}^P.$$

The invertible matrices $\mathbf{M}_L$ and $\mathbf{M}_R$ are called left respectively right preconditioner. Solving linear systems with the preconditioners should be much cheaper than with $\mathbf{A}$; often the preconditioners are triangular matrices. In cases where it is clear a priori what properties slow down convergence, some preconditioners directly modify these properties. For example with symmetric positive definite systems, preconditioners may change unfavorable eigenvalues (see, e.g., [9]). For the nonsymmetric systems involved in our applications convergence hampering properties are often much harder to cure or even to detect. Therefore, one frequently chooses the preconditioners such that they immediately approach $\mathbf{A}$. One usually preconditions either from the right or from the left, where left preconditioning has the advantage no final substitution to obtain $\mathbf{x}$ from $\mathbf{x^P}$ is needed. On the other hand, with zero initial guesses right preconditioning does not artificially modify initial residual norms. From experience, right preconditioning seems to be the better choice in the context of compressible flows. Therefore, from now on we assume $\mathbf{M}$ is a right preconditioner approximating $\mathbf{A}$ which is applied as

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{x}^P = \mathbf{b}, \quad \mathbf{x} = \mathbf{M}^{-1}\mathbf{x}^P.$$

For a classical introduction into preconditioners we refer to [1]. An overview of preconditioners with special emphasis on application in flow problems can be found in [13] and the study in the context of compressible flow [16]. In our context, the most appropriate class of preconditioners is that of incomplete LU (ILU) decompositions. The application of such a decomposition as a preconditioner corresponds to forward and backward substitution with the factors $\mathbf{L}$ and $\mathbf{U}$, respectively. The computation of an incomplete LU decomposition is derived from the Gaussian elimination process for full decompositions. For example, during the process, one prescribes a certain sparsity pattern of the triangular factors. The sparsity pattern may be influenced by the level of fill. This is in short a measure for how much fill beyond the original sparsity pattern is allowed for the purpose of an ILU decomposition. Another idea is to use a drop tolerance and neglect all values in the incomplete factorization below a threshold value. Of course, the two approaches can be combined as in the ILUT preconditioners from [20]. We will here concentrate on ILU(0), which has no additional level of fill beyond the sparsity pattern of the original matrix $\mathbf{A}$. This has the obvious advantage that it enables straightforward a priori allocation. Though ILU(0) may not be powerful enough for some difficult problems, for an important number of applications from CFD, including our model problems, it is efficient. In fact, as most problems have a block structure, the used preconditioner is a *block* ILU(0) decomposition (BILU(0)) where in the Gaussian elimination process pointwise operations are replaced by blockwise operations. In our model problems, the blocks correspond to the 4×4 units the Jacobian consists of (see (4)). For the involved BILU(0) decompositions we use the following notation. We assume they are computed rowwise, hence the result is a block lower triangular factor denoted by $\mathbf{L}$ with 4×4 identity matrices on the main diagonal and a block upper triangular factor

$\mathbf{U_D}$ with arbitrary nonsingular 4×4 matrices on the main diagonal. In addition, we denote by $\mathbf{D}$ the block diagonal part of $\mathbf{U_D}$ and let $\mathbf{U}$ be the matrix $\mathbf{U_D}$ scaled by $\mathbf{D^{-1}}$, i.e. $\mathbf{U} = \mathbf{D^{-1}U_D}$. Then $\mathbf{U}$ has, like $\mathbf{L}$, 4×4 identity matrices on its main diagonal.

The main focus of this paper is efficient preconditioning of the *sequences* of linear systems arising from the scheme described above. Some strategies to share part of the computational effort throughout a sequence were mentioned in the introduction. The two tools we will use here are *periodic recomputation* of preconditioners combined with *approximate updating*. The idea of periodic recomputation is clear: Computing the preconditioner for every new linear system is time-consuming and unnecessary when the system matrices change slowly. Therefore, we will freeze preconditioners while solving several subsequent systems. We will not consider here the problem of finding optimal recomputation periods. Instead, we concentrate on a straightforward way to update the frozen preconditioners, thus enhancing their power. The technique we base our updates on is described in [7]. In the next section we have reformulated this strategy for the type of decomposition used here. We present several mathematical statements on the efficiency of the updates when using BILU(0) preconditioning. Furthermore, we give a detailed description of implementational aspects relevant while applying the updates in our applications.

3.2 Preconditioner Updates

In addition to a system $\mathbf{Ax} = \mathbf{b}$ with preconditioner $\mathbf{M} = \mathbf{LU_D} = \mathbf{LDU}$, let $\mathbf{A^+x^+} = \mathbf{b^+}$ be a system of the same dimension arising later in the sequence and denote the difference matrix $\mathbf{A} - \mathbf{A^+}$ by $\mathbf{B}$. We search for an updated preconditioner $\mathbf{M^+}$ for $\mathbf{A^+x^+} = \mathbf{b^+}$. We have

$$\|\mathbf{A} - \mathbf{M}\| = \|\mathbf{A^+} - (\mathbf{M} - \mathbf{B})\|,$$

hence the level of accuracy of $\mathbf{M^+} \equiv \mathbf{M} - \mathbf{B}$ for $\mathbf{A^+}$ is the same, in the chosen norm, as that of $\mathbf{M}$ for $\mathbf{A}$. The update techniques from [7] are based on the *ideal* updated preconditioner $\mathbf{M^+} = \mathbf{M} - \mathbf{B}$. If we would use it as a preconditioner, we would need to solve systems with $\mathbf{M} - \mathbf{B}$ as system matrix in every iteration of the linear solver. Clearly, for general difference matrices $\mathbf{B}$ the ideal updated preconditioner cannot be used in practice since the systems would be too hard to solve. We will consider cheap approximations of $\mathbf{M} - \mathbf{B}$ instead.

If $\mathbf{M} - \mathbf{B}$ is nonsingular, we approximate its inverse by a product of factors which are easier to invert. The approximation consists of two steps. First, we approximate $\mathbf{M} - \mathbf{B}$ as

$$\mathbf{M} - \mathbf{B} = \mathbf{L}(\mathbf{U_D} - \mathbf{L^{-1}B}) \approx \mathbf{L}(\mathbf{U_D} - \mathbf{B}), \tag{5}$$

or by

$$\mathbf{M} - \mathbf{B} = (\mathbf{LD} - \mathbf{BU^{-1}})\mathbf{U} \approx (\mathbf{LD} - \mathbf{B})\mathbf{U}. \tag{6}$$

Next we replace $\mathbf{U_D} - \mathbf{B}$ or $\mathbf{LD} - \mathbf{B}$ by a nonsingular and easily invertible approximation. In [7] several options are proposed. We have here modified the first option in order to apply it to BILU(0) preconditioners and will approximate as

$$\mathbf{U_D} - \mathbf{B} \approx \mathit{btriu}(\mathbf{U_D} - \mathbf{B}),$$

or as

$$\mathbf{LD} - \mathbf{B} \approx btril(\mathbf{LD} - \mathbf{B}),$$

where *btriu* and *btril* denote the block upper and block lower triangular parts (including the main diagonal), respectively. Putting the two approximation steps together, we obtain updated preconditioners of the form

$$\mathbf{M}^+ = \mathbf{L}(\mathbf{U_D} - btriu(\mathbf{B})) \tag{7}$$

and

$$\mathbf{M}^+ = (\mathbf{LD} - btril(\mathbf{B}))\mathbf{U}. \tag{8}$$

They can be obtained with very low costs. They ask only for subtracting block triangular parts of $\mathbf{A}$ and $\mathbf{A}^+$ (and for saving the corresponding block triangular part of $\mathbf{A}$). In addition, as the sparsity patterns of the factors from the BILU(0) factorization and from the block triangular parts of $\mathbf{A}$ (and $\mathbf{A}^+$) are identical, both backward and forward substitution with the updated preconditioners are as cheap as with the frozen preconditioner $\mathbf{LU_D} = \mathbf{LDU}$.

It is clear from the two approximations we make, that the distance of the updated preconditioners to the ideal preconditioner is influenced by mainly two properties. The first is closeness of $\mathbf{L}$ or $\mathbf{U}$ to the identity. If matrices have a strong diagonal, the diagonal dominance is in general inherited by the factors $\mathbf{L}$ and $\mathbf{U}$ [4, 3], yielding reasonable approximations of the identity. Note that the form (4) of matrices from compressible flow problems admits easy detection of diagonal dominance. The second property that helps in approximating the ideal preconditioner is a block triangular part containing significantly more relevant information than the other part. One may think, for example, of upwind discretization schemes where difference matrices are fully (block) upper triangular. In one of our model problems we emphasize one triangular part by using a numbering of grid cells corresponding to the direction of the flow characteristics. Summarizing, one may expect updates of the form (7) or (8) to be accurate whenever $btril(\mathbf{B})$ or $btriu(\mathbf{B})$ is a useful approximation of $\mathbf{B}$ and when the factor $\mathbf{L}$ or $\mathbf{U}$ is close to the identity matrix. The following lemma suggests that under the mentioned circumstances, the updates have the potential to be more accurate than the frozen or any other (possibly recomputed) preconditioner for $\mathbf{A}^+$.

Lemma 1 *Let* $||\mathbf{A} - \mathbf{LDU}|| = \varepsilon||\mathbf{A}|| < ||\mathbf{B}||$ *for some* $\varepsilon > 0$. *Then the preconditioner from (8) satisfies*

$$||\mathbf{A}^+ - \mathbf{M}^+|| \quad \leq \quad \frac{\|\mathbf{U}\|\,\|bstriu(\mathbf{B})\| + ||\mathbf{U} - \mathbf{I}||\,\|\mathbf{B}\| + \varepsilon||\mathbf{A}||}{||\mathbf{B}|| - \varepsilon||\mathbf{A}||} \cdot ||\mathbf{A}^+ - \mathbf{LDU}||,$$

where bstriu denotes the block strict upper triangular part.

This result is a straightforward modification of Lemma 2.1 in [7]; a similar statement can be obtained for updates of the form (7). The next result is more specific for the situation we are interested in here. It exploits the fact that the BILU(0) preconditioner is an exact decomposition on the sparsity pattern of the matrix it preconditions. The matrix $\mathbf{E}$ denotes the error $\mathbf{E} \equiv \mathbf{A} - \mathbf{LDU}$ of the BILU(0) preconditioner.

Lemma 2 *Let*

$$\rho = \frac{\|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F \left(2 \cdot \|\mathbf{E} - bstriu(\mathbf{B})\|_F + \|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F\right)}{\|btril(\mathbf{B})\|_F^2} < 1,$$

where bstriu denotes the block strict upper triangular part of a matrix. Then the accuracy $\|\mathbf{A}^+ - (\mathbf{LD}-btril(\mathbf{B}))\mathbf{U}\|_F$ *of the updated preconditioner (8) is higher than the accuracy of the frozen preconditioner* $\|\mathbf{A}^+ - \mathbf{LDU}\|_F^2$ *with*

$$\|\mathbf{A}^+ - (\mathbf{LD}-btril(\mathbf{B}))\mathbf{U}\|_F \leq \sqrt{\|\mathbf{A}^+ - \mathbf{LDU}\|_F^2 - (1-\rho)\|btril(\mathbf{B})\|_F^2}. \tag{9}$$

P r o o f : We have, by assumption,

$$\begin{aligned}
\|\mathbf{A}^+ - (\mathbf{LD}-btril(\mathbf{B}))\mathbf{U}\|_F^2 &= \|\mathbf{A} - \mathbf{LDU} - \mathbf{B}+btril(\mathbf{B})\mathbf{U}\|_F^2 \\
&= \|\mathbf{E}-bstriu(\mathbf{B}) + btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F^2 \\
&\leq (\|\mathbf{E} - bstriu(\mathbf{B})\|_F + \|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F)^2 \\
&= \|\mathbf{E} - bstriu(\mathbf{B})\|_F^2 + \rho\|btril(\mathbf{B})\|_F^2.
\end{aligned}$$

Note that the sparsity patterns of $\mathbf{A}$ and $\mathbf{E}$ are disjoint. Therefore,

$$\|\mathbf{E} - bstriu(\mathbf{B})\|_F^2 + \|btril(\mathbf{B})\|_F^2 = \|\mathbf{E}\|_F^2 + \|\mathbf{B}\|_F^2 = \|\mathbf{E}-\mathbf{B}\|_F^2 = \|\mathbf{A}^+ - \mathbf{LDU}\|_F^2.$$

Hence

$$\|\mathbf{E}-bstriu(\mathbf{B})\|_F^2 + \rho\|btril(\mathbf{B})\|_F^2 = \|\mathbf{A}^+ - \mathbf{LDU}\|_F^2 - (1-\rho)\|btril(\mathbf{B})\|_F^2.$$

□

In a nutshell, the superiority of the updated over the frozen preconditioner is mainly determined by the ratio

$$\frac{\|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F}{\|btril(\mathbf{B})\|_F}. \tag{10}$$

We will see in our experiments that ρ (and hence this ratio) tends to be much smaller than one.

In addition to the accuracy $\|\mathbf{A}^+ - \mathbf{M}^+\|$ also norms of $\mathbf{I} - \mathbf{A}^+(\mathbf{M}^+)^{-1}$ provide information on the quality of the update. They indicate more about the *stability* of the preconditioner and may have an important impact on the number of iterations needed to solve a linear system as well [**?**]. For example, if a triangular factor of $\mathbf{M}^+$ has strong off-diagonal but weak diagonal entries, then back- or forward substitution with the factor will be badly conditioned, thus risking to yield severe delay of convergence. The following theoretical result expresses the matrix $\mathbf{I} - \mathbf{A}^+(\mathbf{M}^+)^{-1}$ for updates of the form (7) as the sum of a low rank matrix and a matrix whose norm can be bounded from above. The latter norm may be small with a good approximation $btriu(\mathbf{B})$ of $\mathbf{B}$ and a factor $\mathbf{L}$ reasonably close to the identity. In exact arithmetics, Krylov subspace methods converge in t steps when the system matrix is the identity plus a rank t matrix.

Theorem 3 *Let there exist a number* $c \in \mathbb{R}$ *with*

$$\|\mathbf{U_D^{-1}}\, btriu(\mathbf{B})\|_2 \leq 1/c < 1$$

where $\|\cdot\|_2$ *denotes the Euclidean norm. Further assume that the singular values* σ_i *of*

$$\mathbf{B - E - L}\, btriu(\mathbf{B})$$

satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_t \geq \delta \geq \sigma_{t+1} \geq \cdots \geq \sigma_N$ *for some integer* t, $t \ll N$, *and some small* $\delta > 0$. *Let the block diagonal part* $\mathbf{D}$ *of* $\mathbf{U_D}$ *have the form* $\mathbf{D} = \text{blockdiag}(\mathbf{D_1}, \ldots, \mathbf{D_N})$. *Then there exist matrices* $\mathbf{F}$ *and* $\mathbf{\Delta}$ *such that*

$$\mathbf{A}^+ \left(\mathbf{U_D} - btriu(\mathbf{B})\right)^{-1} \mathbf{L^{-1}} = \mathbf{I} + \mathbf{\Delta} + \mathbf{F},$$

with $rank(\mathbf{\Delta}) \leq t$ *and*

$$\|\mathbf{F}\|_\mathbf{2} \leq \frac{c}{c-1} \max_i \frac{\delta}{\|\mathbf{D_i}\|_2} \|\mathbf{L^{-1}}\|_\mathbf{2} \|\mathbf{U^{-1}}\|_2.$$

P r o o f: Consider Theorem 2.2 in [7]. With $\overline{\mathbf{U_D - B}} \equiv btriu(\mathbf{U_D - B})$ the matrix

$$(\mathbf{I - L})\mathbf{B} + \mathbf{L}\left(\overline{\mathbf{U_D - B}} - (\mathbf{U_D} + \mathbf{L^{-1}E - B})\right)$$

takes the form $\mathbf{B - E - L}\, btriu(\mathbf{B})$ and $\overline{\mathbf{U^{-1}D^{-1}B}}$ translates to $\mathbf{U_D^{-1}}\, btriu(\mathbf{B})$. Preconditioning from the right instead of from the left and blockwise formulation have no influence on the derived results. □

We will now describe how we exploit updated preconditioners of the form (7) and (8) in the solution process of the problems introduced in previous sections. A first issue is choosing between (7) and (8). Just as we approximate the ideal preconditioner in two steps, there are basically two types of criteria that can be used. The first criterion compares the closeness of $\mathbf{U}$ and $\mathbf{L}$ to identity. If $\mathbf{L}$ is closer, then we assume the approximation made in (5) is better than in (6) and we prefer to update the upper triangular part of the decomposition as given in (7); if $\mathbf{U}$ is closer to identity, we update the lower triangular part according to (8). Note that a factor close to identity also leads to stable back- or forward substitution with the factor. Therefore, an important consequence of choosing the factor which is closest to identity is that we keep, in the update, the stablest part of the initial decomposition (and thus we may positively influence the first assumption in Theorem 3). Due to the lack of diagonal dominance in our applications, stability of the factors is a relevant issue. We call this criterion the *stable update criterion.* On the other hand, it is clear that the quality of the approximation $\mathbf{U_D} - btriu(\mathbf{B})$ of $\mathbf{U_D - B}$ (or $\mathbf{LD} - btril(\mathbf{B})$ of $\mathbf{LD - B}$) may have a decisive influence on the power of the preconditioner. The second criterion consists of comparing $\|btril(\mathbf{B})\|$ and $\|btriu(\mathbf{B})\|$. We assume the most important information is contained in the dominating block triangular part and therefore we update with (7) if $btriu(\mathbf{B})$ dominates and otherwise with (8). This rule is denoted by *information flow criterion.* Note that in our implementation we always used the Frobenius norm to evaluate the criteria.

Our model problems lead to systems with a block structure and for efficiency reasons, this block structure should be exploited whenever possible. In order to solve linear systems

blockwise and, in particular, work with BILU(0) decompositions, we have adapted the original updating technique to updates of the form (7) and (8). Blockwise decompositions, however, make switching between (7) and (8) a little more complicated than with classical pointwise decompositions. Using the update (7) is straightforward but note that in order to obtain $\mathbf{U}$ and to apply (8) we need to scale $\mathbf{U_D}$ by $\mathbf{D^{-1}}$, as we explained in Section 3.1. Scaling with inverse block diagonal matrices does have, in contrast with inverse diagonal matrices, some influence on overall performance and should be avoided as much as possible. Note that our stable update criterion compares factors $\mathbf{L}$ and $\mathbf{U}$, both with block diagonal consisting of identity blocks. This means that in order to use the criterion we need to scale $\mathbf{U_D}$, even if the criterion decides for (7) and scaling would not have been necessary. We may circumvent this possible inefficiency by considering $\mathbf{U_D}$ and $\mathbf{LD}$ instead of $\mathbf{U}$ and $\mathbf{L}$. More precisely, we would compare $\|\mathbf{D} - \mathbf{U_D}\|$ with $\|\mathbf{LD} - \mathbf{D}\|$. We call this criterion the *unscaled stable update criterion*.

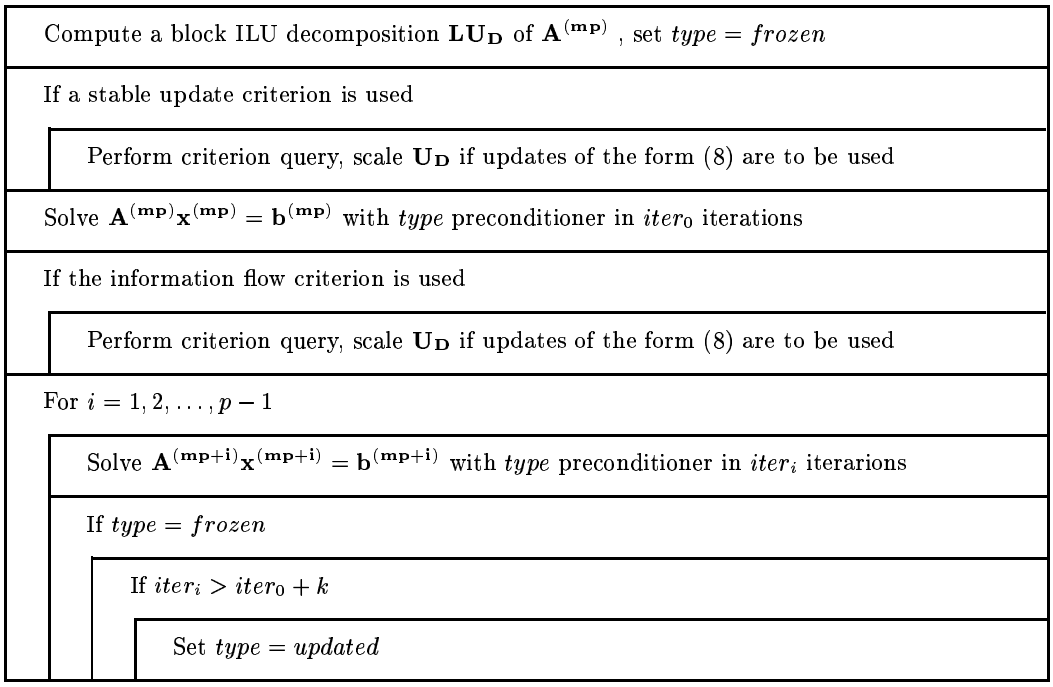
A related issue is the frequency of deciding about the update type based on a criterion. On the one hand, there may be important differences in the performance of (7) and (8); on the other hand, switching between the two types can imply some additional costs as we explained in the previous paragraph and it is implementationally complicated. We believe that the criterion query should not be repeated too often. We adopted the following strategy. After every recomputation of a block ILU-decomposition, which takes place periodically, we perform one query and then use the chosen type of update throughout the whole period. With the information flow criterion we compare $\|btril(\mathbf{B})\|$ with $\|btriu(\mathbf{B})\|$ for the first difference matrix $\mathbf{B}$ generated after recomputation, i.e. just before solving the system following the system for which we computed a new block ILU-decomposition. With the stable update criteria we may decide immediately after the new block ILU-decomposition was computed upon what update type to use in the period to come. Note that as soon as an update type is chosen, we need to store only one triangular part of the old reference matrix $\mathbf{A}$ (and two triangular factors of the reference decomposition).

Another property of the applications we are interested in here, is that the solution process typically contains heavily instationary phases followed by long nearly stationary phases. This is reflected by parts of the sequence of linear systems with large entries in the difference matrices and other parts where system matrices are very close. Obviously, in the latter parts we may expect a frozen preconditioner to be powerful for many subsequent systems. Our experiments confirm this: In stationary phases we typically observe a deterioration of only 2 to 5 iterations with respect to the iterations needed to solve the system for which the frozen preconditioner was computed. Updating the frozen preconditioner in these cases would be counterproductive; it would add some overhead which cannot be compensated by the few savings of iterations. In fact, in these cases there is even a risk that updates produce more iterations, especially when the frozen preconditioner is particularly stable. We therefore apply a very simple technique to avoid unnecessary updating. We start every period by freezing the preconditioner. Denote the number of iterations of the linear solver needed to solve the first system of the period by $iter_0$. If for the $(j+1)$st system the corresponding number of iterations $iter_j$ satisfies

$$iter_j > iter_0 + k, \tag{11}$$

with some threshold $k \in \mathbb{N}$, then we use updates for all remaining systems of the period. In accordance with our observations, in practice we use $k \in \{2, 3, 4, 5\}$. To get a clearer

Flow diagram — preconditioner update decisions after every recomputation



impression of the many decisions to be made we have added a flow diagram. Here, p denotes the recomputation period and $m = 0, 1, 2, \ldots$.

4 Numerical Experiments

In this section we demonstrate the behavior of the update technique on some well known steady test cases. The corresponding linear equation systems are solved until the initial residual has dropped by a factor of 10^7. We always compare periodic refactorization without updating to periodic refactorization with updating, where also the 3 criteria for deciding whether to use upper or lower updating are compared. The total number of BiCGSTAB iterations as well as the total CPU time for the whole run are recorded. Our primary indicator to evaluate performance is the CPU time, as a small number of BiCGSTAB iterations may be due to a powerful preconditioner that takes tremendous amount of computing time. All computations were done on a Pentium IV with 2.4 GHz.

4.1 Supersonic flow past a cylinder

The first model problem is frontal flow at Mach 10 around a cylinder, which leads to a steady state. 3000 steps of the implicit Euler method are performed. The grid consists of 20994 points, whereby only a quarter of the domain is discretized, and system matrices are of dimension 83976. The number of nonzeroes is about $1.33 \cdot 10^6$ for all matrices of the

sequence. For the initial data, freestream conditions are used. Thus, in the beginning, a strong shock detaches from the cylinder, which then slowly moves backward through the domain until reaching the steady state position. Therefore, the linear systems are changing only very slowly during the last 2500 time steps and all important changes take place in the initial phase of 500 time steps. The initial CFL number is 5, which is increased up to 7 during the iteration. The solution is shown in figure 2.

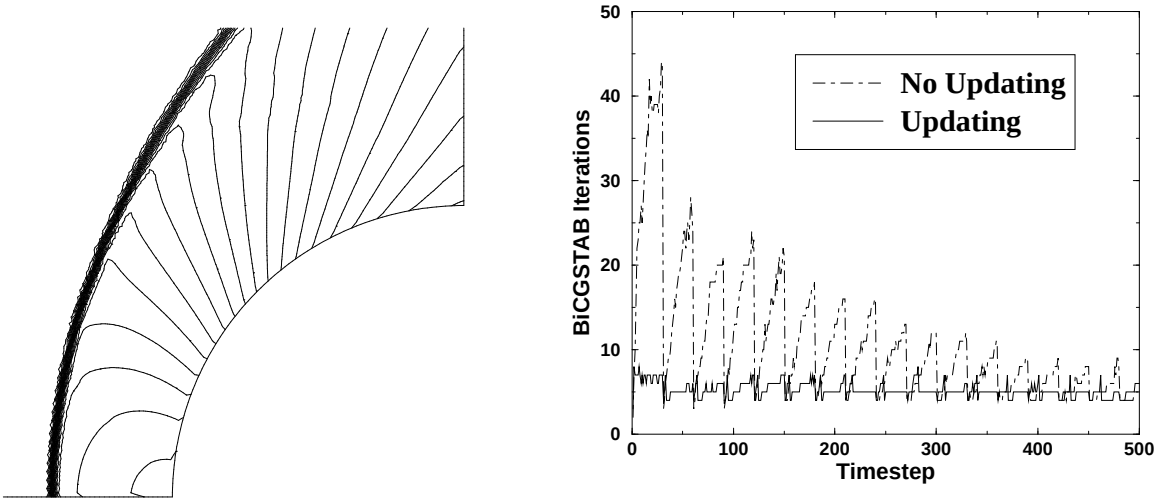


Figure 2: Pressure Isolines (left) and BiCGSTAB iterations per time step (right) for the Cylinder problem

As the flow is supersonic, the characteristics point mostly in one direction. The performance of the linear equation solver can be improved by choosing a numbering of the grid cells that respects the direction of the flow, thereby making the matrix more triangular in nature [16]. Renumbering reduces the total number of BiCGSTAB iterations by about thirty percent. Furthermore, dominance of one of the two triangular parts is exactly the situation in which we expect the update technique to work well. Recall that Lemma 1 (and in a less explicit way Lemma 2 and Theorem 3) suggests that the updated preconditioner is favorably influenced by matrices that are close to triangular. In figure 2 excellent performance of the updates is shown for the initial insteady phase of the first 500 time steps. As subsequent linear systems change heavily, frozen preconditioners produce rapidly deteriorating numbers of BiCGSTAB iterations (with decreasing peaks demonstrating the convergence to steady state). Updating, on the other hand, yields a nearly constant number of iterations per time step. The recomputing period here is thirty and the criterion used is the stable update criterion but other periods and criteria give a similar result. With freezing, 5380 BiCGSTAB iterations are performed in this part of the solution process, while the same computation with updating needs only 2611 iterations.

In the following table we explain the superior performance of the updates with the quantities from Lemma 2 for the very first time steps; they demonstrate the general trend for the whole instationary phase. Here, $M^{(i)}$ denotes the update (8) for the ith linear system. As the upper bound (9) on the accuracy of the updates is very tight, we conclude that in this problem the power of the updates is essentially due to the small values of ρ, i.e. of the

ratio (10):

i	$\|A^{(i)} - LDU\|_F$	$\|A^{(i)} - M^{(i)}\|_F$	Bound from (9)	ρ from (9)
2	37.454	34.277	36.172	0.571
3	37.815	34.475	36.411	0.551
4	42.096	34.959	36.938	0.245
5	50.965	35.517	37.557	0.104
6	55.902	36.118	38.308	0.083

In the next table we display the performance of the updates for the whole sequence:

	No updating		Stable update		Unscaled stable update		Information flow	
Period	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s
10	10683	7020	11782	7284	11782	7443	11782	7309
20	12294	6340	12147	6163	12147	6300	12147	6276
30	13787	7119	12503	5886	12503	5894	12503	5991
40	15165	6356	12916	5866	12916	5835	12916	5856
50	16569	6709	11962	5821	13139	5670	13139	5925

To evaluate the results, first note that the reduction of the BiCGSTAB iterations happens primarily in the first 500 time steps. After 500 time steps, freezing is a very efficient strategy and actually gains again on updating. Thus the success of updating is somewhat damped by the long stationary tail of this model problem. The different updating strategies lead to nearly identical results, whereby the stable update criterion is the best. As expected, the update criterions all choose to update the lower triangular part according to (8), as the upper triangular part is close to identity due to the numbering of the unknowns and the high Mach number. Therefore, they all obtain the same iteration numbers (except for one case). Updating is clearly better than freezing if the recomputing period is at least 20. For recomputing periods of 30 or greater, the performance of the updating strategy does not much depend on the period. The CPU time is decreased by about 10 % in general; with the recomputing period 50 it reaches up to 20 %. For high recomputing periods, the number of iterations is reduced by even more than 20 %. If the ILU decomposition would have been recomputed in every step, only 11099 BiCGSTAB iterations would be needed, but 28583 seconds of CPU time.

4.2 Flow past a NACA0012 airfoil

The second model problem corresponds to the NACA0012 profile at an angle of attack of two degrees on a grid with 4605 cells at different Mach numbers. System matrices are of dimension 18420 and the number of non-zeroes is about $5{\cdot}10^5$ for all matrices of the sequence. For the initial data, freestream conditions are used.

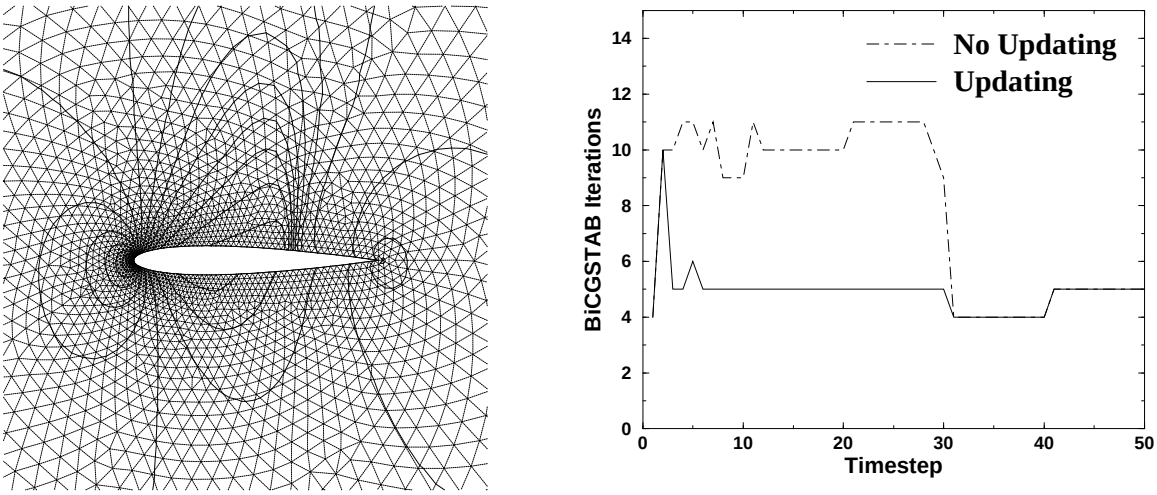


Figure 3: Pressure Isolines and grid (left) and BiCGSTAB iterations per time step (right) for NACA profile with Mach 0.8

At first we consider a reference Mach number of $M = 0.8$. The initial CFL number is 5, which is increased up to 30 during the process. For the solution, see figure 3 (left). Transition to steady state is such that at first a shock above the airfoil is formed, after which the rate of convergence slows down, even though the CFL number is increased. Similarly as in the supersonic model, the equation systems differ much from step to step at first, but are very close towards the end. In fact, this behavior is here even more extreme: With decisions based on (11), updating is applied during the very first period only. To illustrate this, figure 3 (right) compares, for recomputation with a period of 30 time steps, classical freezing with our strategy. Clearly, increasing BiCGSTAB iteration numbers of the frozen preconditioner can be corrected with the updates. But after the first period, there is no need to correct anymore. For the entire process we obtain the following table:

	No updating		Stable update		Unscaled stable update		Information flow	
Period	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s
10	5375	543	5336	498	5336	494	5336	483
20	5454	497	5364	469	5364	468	5364	459
30	5526	491	5379	464	5379	467	5379	453
40	5558	491	5411	456	5411	462	5411	452
50	5643	525	5413	466	5413	470	5413	448

As we can see in the tabular, the number of iterations decreases if the recomputation period is shortened. This is not true for the CPU time, as recomputations are costly. For the strategy without updating, the CPU time decreases at first, but increases again, as the benefit of fewer recomputations is balanced by the increase in BiCGSTAB iterations. As for the different updating strategies, all lead to both fewer iterations and shorter computing times. As we explained before, the reduction of iterations must be solely due to the very first time steps where updates are applied. The information flow criterion is the fastest,

whereas the stable update criterion and the unscaled stable update criterion are somewhat slower, but still faster than without updating. All three criterions lead to an identical number of BiCGSTAB iterations, because they always choose the same triangular part to update. If the ILU decomposition would have been recomputed in every step, only 5333 BiCGSTAB-iterations would be needed, but 964 seconds of CPU time. Thus the number of iterations with updating often comes close to the number with refactorization in every single step. The differences in CPU time come from the cost of selecting the appropriate triangular part and all in all, the computation of the steady state is sped up by about 7 to 15%. Note that again, the CPU time depends less on the choice of the recomputation period with updates than is the case without updating.

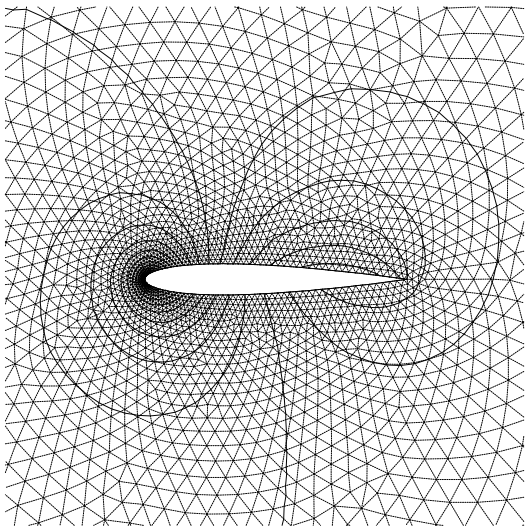


Figure 4: Pressure Isolines and Grid for NACA profile at Mach 0.001.

In the last test case we use a Mach number of $M = 0.001$. This problem is much more stiff than the transonic problem and the linear systems are harder to solve. Furthermore, for the same CFL number, the time steps are much smaller due to the larger maximal eigenvalue. We computed 750 time steps, starting with a CFL number of 0.5, which was increased to 2 over time. For the solution, see figure 4, for the comparison of updating techniques see the last table. In this case, the linear systems do not differ much from time step to time step, not even in the beginning. Thus, the freezing strategy works well and the number of iterations needed increases very slowly in one recomputation cycle. Therefore, even if updating is used, the criterion (11) is seldom fulfilled and the updating strategy has only a small effect in decreasing the iteration numbers, but essentially none on the CPU time. Nevertheless, it is not worse than the classic strategy, which is mainly due to the inclusion of criterion (11): Otherwise, the method would compute an update in every step to no effect. Note that if the ILU decomposition would have been recomputed in every step, 19609 BiCGSTAB-iterations would be needed, but 1437 seconds of CPU time.

	No updating		Stable update		Unscaled stable update		Information flow	
Period	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s
10	19444	1189	19288	1158	19398	1121	19289	1129
20	19584	1105	19492	1135	19451	1117	19375	1094
30	19641	1144	19412	1122	19531	1158	19544	1112
40	19622	1104	19521	1112	19594	1114	19523	1107
50	19622	1127	19265	1129	19339	1086	19396	1139

5 Conclusions

We employed an updating method for block ILU preconditioners for sequences of non-symmetric linear systems in the context of compressible flow. The updating method was motivated by the need to improve frozen preconditioners in order to obtain preconditioners similarly powerful as if they would have been recomputed. For the model problems considered here we showed that as soon as frozen preconditioners yield high numbers of iterations of the linear solver, the updates indeed succeed in reducing the number to the normal level. Whereas the derivation of the updates assumes diagonal dominance of system matrices, the present experiments imply the technique is efficient with rather poor diagonal dominance as well.

Based on the number of Krylov subspace method iterations, our implementation decides whether updating is necessary. In this way we obtained a preconditioning strategy that is faster than the standard strategy of periodic recomputing for well known test cases and it is even close to recomputing in every step with respect to iteration numbers. In contrast with periodic recomputation without updates, our method is rather insensitive to the chosen recomputation period. The method is particularly successful in the phases where the solution process exhibits some kind of instationary behavior and thus it is promising for the computation of instationary flows. In our tables we willingly chose to display results for the whole solution process including stationary phases; when restricted to the phases where updating is actually applied the results would look even more convincing.

References

[1] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.

[2] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel, *Adaptively preconditioned GMRES algorithms*, SIAM J. Sci. Comput., 20 (1998), pp. 243–269.

[3] M. Benzi and D. Bertaccini, *Approximate inverse preconditioning for shifted linear systems*, BIT, 43 (2003), pp. 231–244.

[4] M. Benzi and M. Tůma, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1851–1868.

[5] L. Bergamaschi, R. Bru, A. Martínez, and M. Putti, *Quasi-Newton preconditioners for the inexact Newton method*, ETNA, 23 (2006), pp. 76–87.

[6] D. Bertaccini, *Efficient preconditioning for sequences of parametric complex symmetric linear systems*, Electronic Transactions on Numerical Mathematics, 18 (2004), pp. 49–64.

[7] J. Duintjer Tebbens and M. Tůma, *Efficient preconditioning of sequences of nonsymmetric linear systems*, SIAM J. Sci. Comput., to appear (2007).

[8] H. C. Elman and A. Ramage, *Fourier analysis of multigrid for the two-dimensional convection-diffusion equation*, BIT Numer. Math., online version (May, 2006).

[9] J. Frank and C. Vuik, *On the construction of deflation-based preconditioners*, SIAM J. Sci. Comput., 23 (2001), pp. 442–462.

[10] D. Kröner, *Numerical schemes for conservation laws*, Wiley-Teubner Series Advances in Numerical Mathematics, John Wiley & Sons Ltd., Chichester, 1997.

[11] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, Cambridge Texts in Applied Mathematics, Cambridge University Press, Cambridge, 2002.

[12] D. Loghin, D. Ruiz, and A. Touhami, *Adaptive preconditioners for nonlinear systems of equations*, J. Comput. Appl. Math., 189 (2006), pp. 326–374.

[13] A. Meister, *Numerik linearer Gleichungssysteme*, Friedr. Vieweg & Sohn, Braunschweig, 1999. Eine Einführung in moderne Verfahren. [An introduction to modern procedures].

[14] ——, *Asymptotic based preconditioning technique for low mach number flows*, Z. Angew. Math. Mech., 83 (2003), pp. 3–25.

[15] A. Meister and T. Sonar, *Finite-volume schemes for compressible fluid flow*, Surveys Math. Indust., 8 (1998), pp. 1–36.

[16] A. Meister and C. Vömel, *Efficient preconditioning of linear systems arising from the discretization of hyperbolic conservation laws*, Adv. Comput. Math., 14 (2001), pp. 49–73.

[17] J. Morales and J. Nocedal, *Automatic preconditioning by limited-memory quasi-Newton updates*, SIAM J. Opt., 10 (2000), pp. 1079–1096.

[18] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti, *Recycling Krylov subspaces for sequences of linear systems*, Technical Report UIUCDCS-R-2004-2421, University of Illinois, 2004.

[19] ——, *Recycling Krylov subspaces for sequences of linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 1651–1674.

[20] Y. Saad, *ILUT: a dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.

[21] ——, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, second ed., 2003.

[22] Y. Saad and M. H. Schulz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[23] V. V. Shaĭdurov, *Multigrid methods for finite elements*, vol. 318 of Mathematics and its Applications, Kluwer Academic Publishers Group, Dordrecht, 1995. Translated from the 1989 Russian original by N. B. Urusova and revised by the author.

[24] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems*, SIAM J. Sci. Stat. Comput., 12 (1992), pp. 631–644.

[25] Y. Wada and M.-S. Liou, *A flux splitting scheme with high-resolution and robustness for discontinuities*, AIAA Paper, 94-0083 (1994).

[26] C.-T. Wu and H. C. Elman, *Analysis and comparison of geometric and algebraic multigrid for convection-diffusion equations*, SIAM J. Sci. Comput., 28 (2006), pp. 2208–2228.