INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2013; **71**:1055–1072
Published online 28 June 2012 in Wiley Online Library (wileyonlinelibrary.com/journal/nmf). DOI: 10.1002/fld.3702

Numerical flux functions for Reynolds-averaged Navier–Stokes and $k\omega$ turbulence model computations with a line-preconditioned p-multigrid discontinuous Galerkin solver

Marcel Wallraff[1], Tobias Leicht[1,*,†] and Markus Lange-Hegermann[2]

[1]*German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany*
[2]*RWTH Aachen University, Templergraben 64, 52062 Aachen, Germany*

SUMMARY

We present an eigen-decomposition of the quasi-linear convective flux formulation of the completely coupled Reynolds-averaged Navier–Stokes and $k\omega$ turbulence model equations. Based on these results, we formulate different approximate Riemann solvers that can be used as numerical flux functions in a DG discretization. The effect of the different strategies on the solution accuracy is investigated with numerical examples. The actual computations are performed using a p-multigrid algorithm. To this end, we formulate a framework with a backward-Euler smoother in which the linear systems are solved with a general preconditioned Krylov method. We present matrix-free implementations and memory-lean line-Jacobi preconditioners and compare the effects of some parameter choices. In particular, p-multigrid is found to be less efficient than might be expected from recent findings by other authors. This might be due to the consideration of turbulent flow. Copyright © 2012 John Wiley & Sons, Ltd.

Received 19 January 2012; Revised 4 May 2012; Accepted 10 June 2012

KEY WORDS: discontinuous Galerkin; p-multigrid; turbulent compressible flow; Roe scheme; eigenvalue decomposition

1. INTRODUCTION

The last decade has seen a tremendous interest in high-order numerical methods, in particular in the DG FEM. This method enables an intuitive and convenient treatment of convective terms via approximate Riemann solvers employed as numerical flux functions over element interfaces. This property leads to a large interest in the DG approach for CFD applications in the area of compressible aerodynamic flows. The analysis of turbulent flows employing steady-state computations based on Reynolds-averaged Navier–Stokes (RANS) models might be considered as the work-horse in this field. Nevertheless, DG results are relatively rare for this particular application [1–4]. One of the reasons for this might be the stiffness introduced by both the turbulence model equations and the highly stretched meshes typically used for an efficient resolution of turbulent boundary layers.

These stiffness issues necessitate the use of a strong solver. A rather robust method can be achieved with Newton-like methods. For complex problems, typically some form of continuation has to be introduced, for example, a pseudo-time stepping, which transforms the method into a backward-Euler approach (Section 4.2). This approach is predominantly used for recent results [1–4], but it introduces very high memory requirements, as the full Jacobian of a high-order discretization is stored. The necessity to use efficient preconditioners for iterative solvers of the linear systems often yields a further increase in memory consumption.

*Correspondence to: Tobias Leicht, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany.
†E-mail: Tobias.Leicht@dlr.de

The DG method lends itself to a p-multigrid solution procedure, in which the solution of the original problem is accelerated with the help of coarser problems defined on the same computational mesh, but with a reduced order discretization [5, 6]. In this setting, an efficient smoother is required on each level, that is, for each discretization order. Although explicit time-stepping schemes are very memory efficient, they typically yield unacceptable rates of convergence, in particular for stiff problems. In contrast to that, fully implicit methods suffer from tremendous memory requirements.

A promising strategy is to split the original implicit operator, that is, the full Jacobian, into several parts and keeping some of these while neglecting others [4, 7]. If block entries corresponding to lines of elements are retained, the resulting linear systems are block-tri-diagonal and can thus be solved directly. The lines should be chosen in such a way that strong couplings are retained in the reduced system and thus treated implicitly. Clearly, this is important to resolve mesh anisotropies. In isotropic regions of the mesh, the line-implicit operators can be reduced to element-implicit versions in a canonical way by creating lines consisting of single elements [8]. Several approaches to identify lines have been proposed [7, 8], see Section 4.2 for some details on the current implementation.

After introducing the equations and DG discretization briefly in Section 2, this paper presents several numerical flux functions for the convective part, based on an eigen-decomposition of the fully coupled RANS-$k\omega$ model in Section 3. The effect of the choice of a particular numerical flux function on the accuracy of computations is demonstrated in Section 5. Furthermore, a p-multigrid algorithm is presented in Section 4. Again, numerical results will demonstrate the performance of the algorithm.

Although all presented algorithms have been implemented in an unstructured fashion, the computational experiments are performed only on meshes, which have an underlying block structure. This is due to the fact that it is much simpler to obtain good coarse meshes in the structured case. Furthermore, finer structured meshes can be exploited in order to obtain a curvilinear description, which is important for the accurate approximation of non-straight solid wall boundaries.

2. DG DISCRETIZATION OF THE RANS–$K\omega$ EQUATIONS

The steady-state RANS-$k\omega$ equations [9, 10] can be written in conservative form based on convective and viscous fluxes $\mathcal{F}^c$ and $\mathcal{F}^v$, corresponding to first and second order derivatives, respectively, and additional source terms $\mathcal{S}(\mathbf{u}, \nabla\mathbf{u})$ as

$$\nabla \cdot (\mathcal{F}^c(\mathbf{u}) - \mathcal{F}^v(\mathbf{u}, \nabla\mathbf{u})) - \mathcal{S}(\mathbf{u}, \nabla\mathbf{u}) = 0, \tag{1}$$

with the vector of conservative variables $\mathbf{u} = (\rho, \rho v_1, \rho v_2, \rho v_3, \rho E, \rho k, \rho\omega)^T$, where ρ denotes the density, $\mathbf{v} = (v_1, v_2, v_3)^T$ the velocity vector, E the total energy, k the turbulent kinetic energy and ω the turbulence dissipation rate. The (thermodynamic) pressure p can be computed using the equation of state of a perfect gas, $p = \rho RT$. The gas constant R given by the difference between the specific heat capacities at constant pressure, c_p, and constant volume, c_v, $R = c_p - c_v$, as well as the relation $E = c_v T + \frac{1}{2}\mathbf{v}^2 + k$, where T denotes the temperature:

$$p = (\gamma - 1)\left(\rho E - \frac{\rho}{2}\mathbf{v}^2 - \rho k\right).$$

Here, $\gamma := \frac{c_p}{c_v}$ is assumed to be a constant. Defining an effective pressure $p_{\text{eff}} = p + \frac{2}{3}\rho k$ the convective flux is given by $\mathcal{F}^c(\mathbf{u}) = \left(\mathcal{F}^c_1(\mathbf{u}), \mathcal{F}^c_2(\mathbf{u}), \mathcal{F}^c_3(\mathbf{u})\right)$ with

$$\mathcal{F}^c_i(\mathbf{u}) = \mathbf{u}v_i + (0, \delta_{i1}, \delta_{i2}, \delta_{i3}, v_i, 0, 0)^T\, p_{\text{eff}}, \tag{2}$$

where δ denotes the Kronecker delta symbol. The DG discretization of these equations follows standard techniques based on numerical fluxes for convective part as well as the second scheme of Bassi and Rebay (BR2) for the viscous terms [1, 11]. We note that the transformation of the turbulent dissipation rate ω to $\tilde{\omega} = \ln(\omega)$ does not change the structure of the convective flux.

Let $\mathbf{n} = (n_1, n_2, n_3)^T$ denote a unit normal vector on a face between neighboring elements κ^+ and κ^-, pointing from κ^+ to κ^-.

A numerical flux $\mathcal{H}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n})$ is introduced on faces between neighboring elements κ^+ and κ^- as an approximation to the normal convective flux $\mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n}$ because of the reason that the traces $\mathbf{u}^+$ and $\mathbf{u}^-$ of $\mathbf{u}$, taken from elements κ^+ and κ^-, respectively, differ in general, because of the discontinuous nature of the ansatz functions. $\mathcal{H}$ has to be conservative, $\mathcal{H}(\mathbf{v}, \mathbf{w}, \mathbf{n}) = \mathcal{H}(\mathbf{w}, \mathbf{v}, -\mathbf{n})$, and consistent, $\mathcal{H}(\mathbf{v}, \mathbf{v}, \mathbf{n}) = \mathcal{F}^c(\mathbf{v}) \cdot \mathbf{n}$. In principle, any of the approximate Riemann solvers known from finite volume schemes can be utilized. Some examples are presented in Section 3.

3. NUMERICAL FLUX FUNCTION FOR CONVECTIVE TERMS

Several numerical flux formulations are based on the eigen-decomposition of the normal convective flux Jacobian

$$\mathcal{F}_n^{c\prime}(\mathbf{u}) := \frac{\mathrm{d}(\mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n})}{\mathrm{d}\mathbf{u}}. \tag{3}$$

Explicit expressions for the resulting Jacobian matrix can be found in Appendix A. This Jacobian can be written as

$$\mathcal{F}_n^{c\prime} = T \Lambda T^{-1},$$

where $\Lambda = \mathrm{diag}(\lambda_i)$ is the diagonal matrix of eigenvalues of $\mathcal{F}_n^{c\prime}$ and $T = (w_1, \ldots, w_7)$ is the matrix of eigenvectors w_i corresponding to the eigenvalues λ_i. This eigen-decomposition corresponds to the transformation to a set of characteristic variables, which can be treated via upwinding.

3.1. *Roe flux*

The Roe flux [12] can be regarded as an average of the fluxes evaluated for the left and right states plus a dissipation stabilization of the discontinuities applied to the convective variables. It is given by

$$\mathcal{H}^{\mathrm{Roe}}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) = \frac{1}{2}\left(\mathcal{F}^c(\mathbf{u}^+) \cdot \mathbf{n} + \mathcal{F}^c(\mathbf{u}^-) \cdot \mathbf{n} + T\, A\, T^{-1}(\mathbf{u}^+ - \mathbf{u}^-)\right) \qquad \text{with } A = \mathrm{diag}(\alpha_i) \tag{4}$$

and $\alpha_i = |\lambda_i|$ in the original Roe scheme. However, this introduces no dissipation for characteristic variables whose eigenvalue is zero. Several options for an *entropy fix* to overcome this problem have been proposed. A rather simple technique is given by choosing $\alpha_i = \max(|\lambda_i|, \delta_{\mathrm{ef}} \lambda_{\max})$, where $\lambda_{\max} = \max_i |\lambda_i|$ and $0 \leqslant \delta_{\mathrm{ef}} \leqslant 1$ is a free parameter, typically $\delta_{\mathrm{ef}} = 0.1$ [13]. The original scheme without entropy fix is recovered by $\delta_{\mathrm{ef}} = 0$, whereas $\delta_{\mathrm{ef}} = 1$ yields the local Lax–Friedrichs [14] or Rusanov [15] flux $\mathcal{H}^{\mathrm{LF}}$, which can be written in simpler form as

$$\mathcal{H}^{\mathrm{LF}}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) = \frac{1}{2}\left(\mathcal{F}^c(\mathbf{u}^+) \cdot \mathbf{n} + \mathcal{F}^c(\mathbf{u}^-) \cdot \mathbf{n} + \lambda_{\max}(\mathbf{u}^+ - \mathbf{u}^-)\right). \tag{5}$$

A second choice for the entropy fix is inspired by Harten [16],

$$\alpha_i = \begin{cases} \frac{|\lambda_i|^2 + \delta^2}{2\delta}, & \text{if } |\lambda_i| < \delta \\ |\lambda_i|, & \text{else.} \end{cases} \qquad \text{with } \delta = \delta_{\mathrm{ef}} \lambda_{\max}, \tag{6}$$

which has the advantage of being differentiable at the point $|\lambda_i| = \delta$.

The linearization involved in the determination of T and λ_i is performed at a Roe-averaged state, which is given by $\bar{\rho} = \sqrt{\rho^+ \rho^-}$ and $\bar{x} = \frac{\sqrt{\rho^+} x^+ + \sqrt{\rho^-} x^-}{\sqrt{\rho^+} + \sqrt{\rho^-}}$ for $x = v_i, k, \omega, H$, where $H = E + \frac{p}{\rho}$ denotes the total enthalpy.

3.2. *Vijayasundaram flux*

The Vijayasundaram flux [17] is based on separating the convective flux, expressed in characteristic variables, into components corresponding to incoming and outgoing characteristics. The distinction between these is made based on the sign of the corresponding eigenvalue. Introducing $B^{\pm}(\mathbf{u}) = T\Lambda^{\pm}T^{-1}$ with $\Lambda^{+} = \mathrm{diag}(\max(\lambda_i, 0))$ and $\Lambda^{-} = \mathrm{diag}(\min(\lambda_i, 0))$, as well as the average state $\bar{\mathbf{u}} = \frac{\mathbf{u}^{+}+\mathbf{u}^{-}}{2}$ the Vijayasundaram flux is given by

$$\begin{aligned}
\mathcal{H}^{\mathrm{Vija}}(\mathbf{u}^{+}, \mathbf{u}^{-}, \mathbf{n}) &= B^{+}(\bar{\mathbf{u}})\mathbf{u}^{+} + B^{-}(\bar{\mathbf{u}})\mathbf{u}^{-} \\
&= \left[B^{+}(\bar{\mathbf{u}}) + B^{-}(\bar{\mathbf{u}})\right]\bar{\mathbf{u}} + B^{+}(\bar{\mathbf{u}})\left[\mathbf{u}^{+} - \bar{\mathbf{u}}\right] + B^{-}(\bar{\mathbf{u}})\left[\mathbf{u}^{-} - \bar{\mathbf{u}}\right] \\
&= \mathcal{F}^{c}(\bar{\mathbf{u}})\cdot\mathbf{n} + \frac{1}{2}B^{+}(\bar{\mathbf{u}})\left[\mathbf{u}^{+} - \mathbf{u}^{-}\right] + \frac{1}{2}B^{-}(\bar{\mathbf{u}})\left[\mathbf{u}^{-} - \mathbf{u}^{+}\right] \qquad (7) \\
&= \mathcal{F}^{c}(\bar{\mathbf{u}})\cdot\mathbf{n} + \frac{1}{2}\left[B^{+}(\bar{\mathbf{u}}) - B^{-}(\bar{\mathbf{u}})\right]\left[\mathbf{u}^{+} - \mathbf{u}^{-}\right] \\
&= \mathcal{F}^{c}(\bar{\mathbf{u}})\cdot\mathbf{n} + \frac{1}{2}T|\Lambda|T^{-1}\left[\mathbf{u}^{+} - \mathbf{u}^{-}\right].
\end{aligned}$$

The final form in (7) illustrates the close connection to the Roe flux: The Vijayasundaram flux can be obtained from the Roe flux without entropy fix via exchanging the Roe averaging by a simple arithmetic average and replacing the average of flux evaluated for left and right states by the flux of the average state. As such, the resulting flux does not include a stabilization via an entropy fix and corresponds to pure upwinding, which can be expected to yield robustness problems in the solution process. However, for cases in which the solution does converge, computational results might be superior owing to the addition of only minimal dissipation.

3.3. *Characteristic farfield boundary condition*

At the outer boundaries of the computational domain, farfield boundary conditions are applied. A farfield state $\mathbf{u}^{\infty}$ is defined as the state of the free undisturbed flow. The number of prescribed values at each point of inflow and outflow boundaries should correspond to the number of incoming and outgoing characteristics in each case. To this end, farfield boundaries can be prescribed in a weak form employing the Vijayasundaram flux (7) with the outer state $\mathbf{u}^{-}$ replaced by $\mathbf{u}^{\infty}$. This technique is referred to as a characteristic farfield boundary condition.

3.4. *Eigenvalue decomposition of the flux Jacobian*

The eigenvalues of the convective flux Jacobian $\mathcal{F}_n^{c\,\prime}$ (cf. 3) are $\mathbf{v}\cdot\mathbf{n}$ in multiplicity 5 and $\mathbf{v}\cdot\mathbf{n} \pm a_t$ with a turbulent speed of sound $a_t = \sqrt{a^2 + \frac{2}{3}\gamma k} = \sqrt{\gamma\frac{p_{\mathrm{eff}}}{\rho}}$ based on the effective pressure defined in analogy to the usual speed of sound $a = \sqrt{\gamma\frac{p}{\rho}}$ based on the thermodynamic pressure. The first five eigenvectors of $\mathcal{F}_n^{c\,\prime}$ are calculated as a kernel of $\mathcal{F}_n^{c\,\prime} - \mathbf{v}\cdot\mathbf{n}\,\mathbf{I}$ using the GAUSSian algorithm:

$$\begin{aligned}
w_1 &:= \left(n_1, n_1v_1, n_1v_2 + n_3a_t, n_1v_3 - n_2a_t, \frac{n_1}{2}\mathbf{v}^2 + a_t(n_3v_2 - n_2v_3), 0, 0\right)^T, \\
w_2 &:= \left(n_2, n_2v_1 - n_3a_t, n_2v_2, n_2v_3 + n_1a_t, \frac{n_2}{2}\mathbf{v}^2 + a_t(n_1v_3 - n_3v_1), 0, 0\right)^T, \\
w_3 &:= \left(n_3, n_3v_1 + n_2a_t, n_3v_2 - n_1a_t, n_3v_3, \frac{n_3}{2}\mathbf{v}^2 + a_t(n_2v_1 - n_1v_2), 0, 0\right)^T, \\
w_4 &:= \left(0, 0, 0, 0, \gamma - \frac{5}{3}, \gamma - 1, 0\right)^T \quad \text{and} \quad w_5 := (0, 0, 0, 0, 0, 0, 1)^T.
\end{aligned}$$

Here, w_1, w_2 and w_3 are analogous to the results for the Euler equations without turbulence model [18], with the speed of sound replaced by a_t and two additional zero components for the turbulence model variables.

However, this calculation fails for the remaining two eigenvectors because of coefficient growth over the field $K := \mathbb{Q}(n_1, n_2, n_3, v_1, v_2, v_3, E, k, \gamma, \omega, a_t)$. Instead, we compute the sixth eigenvalue symbolically as the kernel of $\mathcal{F}_n^{c\,\prime} - (a_t + \mathbf{v} \cdot \mathbf{n})\,\mathbf{I}$ over the ring $R := \mathbb{Q}\,[n_1, n_2, n_3, v_1, v_2, v_3, E, k, \gamma, \omega, a_t]$ using the GINV-system [19]:

$$w_6 := \left(1, v_1 + a_t\, n_1, v_2 + a_t\, n_2, v_3 + a_t\, n_3, E + \frac{p + \frac{2}{3}\rho k}{\rho} + a_t\, \mathbf{v} \cdot \mathbf{n}, k, \omega\right)^T$$

This non-zero element w_6 of the kernel of $\mathcal{F}^c - (a_t + \mathbf{v} \cdot \mathbf{n})\,\mathbf{I}$ over R is a non-zero element in the kernel over the quotient field K of R. The latter kernel over K is one-dimensional, so w_6 generates this kernel. We substitute a_t by $-a_t$ in w_6 to obtain the seventh eigenvector

$$w_7 := \left(1, v_1 - a_t\, n_1, v_2 - a_t\, n_2, v_3 - a_t\, n_3, E + \frac{p + \frac{2}{3}\rho k}{\rho} - a_t\, \mathbf{v} \cdot \mathbf{n}, k, \omega\right)^T.$$

4. SOLVER ALGORITHMS

Let $\mathbf{L}_l(\mathbf{u}_l) = \mathbf{f}_l$ be the DG discretization of (1), where l denotes the polynomial degree of the discretization. This is done over a broken Sobolev space $\mathbf{V}_{h,l}$, therefore $\mathbf{u}_l \in \mathbf{V}_{h,l}$.

4.1. Nonlinear p-multigrid

In the p-multigrid, the equations are solved by considering a hierarchy of linear spaces on the same grid

$$\mathbf{V}_{h,p_{\min}} \subset \mathbf{V}_{h,p_{\min}+1} \subset \cdots \subset \mathbf{V}_{h,p-1} \subset \mathbf{V}_{h,p}.$$

On every level $l > p_{\min}$, a smoother $GL_l(\mathbf{u}_l, \mathbf{f}_l)$ is used. At the coarsest level $p_{\min}$, a coarse level solver is applied, for example, from Section 4.2. The solution and the defect between the various approximation levels have to be transferred. The restriction operator for the nonlinear state vector is given by an orthogonal L^2-projection $\hat{I}_l^{l-1} : \mathbf{V}_{h,l} \to \mathbf{V}_{h,l-1}$[5]. Furthermore, the prolongation of the nonlinear state vector is obtained via the natural injection $\hat{I}_{l-1}^{l} : \mathbf{V}_{h,l-1} \to \mathbf{V}_{h,l}$, and a restriction operator for the defect between the various approximation levels is defined by $I_l^{l-1} := \left(\hat{I}_{l-1}^{l}\right)^T$.

The p-multigrid algorithm is stated as follows:

Algorithm 1 p-multigrid algorithm $NMG(\mathbf{f}_l, \mathbf{u}_l, m, l, p_{\min}, \tau)$.

```
1:  if $l = p_{\min}$ then
2:      Solve $\mathbf{L}_{p_{\min}}(\mathbf{u}_{p_{\min}}) = \mathbf{f}_{p_{\min}}$
3:      Set Solution to $\tilde{\mathbf{u}}_{p_{\min}}$.
4:  if $l > p_{\min}$ then
5:      for $i = 1$ to $m$ do
6:          $\mathbf{u}_l := GL_l(\mathbf{u}_l, \mathbf{f}_l)$    /* pre-smoothing */
7:      $\mathbf{u}_{l-1}^0 := \hat{I}_l^{l-1}\mathbf{u}_l$
8:      $\mathbf{f}_{l-1} := \mathbf{f}_{l-1} + I_l^{l-1}(\mathbf{f}_l - \mathbf{L}_l(\mathbf{u}_l)) - (\mathbf{f}_{l-1} - \mathbf{L}_{l-1}(\mathbf{u}_{l-1}^0))$
9:      for $k = 1$ to $\tau$ do
10:         $\mathbf{u}_{l-1}^k := NMG(\mathbf{f}_{l-1}, \mathbf{u}_{l-1}^{k-1}, m, l-1, p_{\min}, \tau)$
11:     $\tilde{\mathbf{u}}_l := \mathbf{u}_l + \hat{I}_{l-1}^{l}(\mathbf{u}_{l-1}^{\tau} - \mathbf{u}_{l-1}^0)$
12:     for $i = 1$ to $m$ do
13:         $\tilde{\mathbf{u}}_l := GL_l(\tilde{\mathbf{u}}_l, \mathbf{f}_l)$    /* post-smoothing */
14: return $\tilde{\mathbf{u}}_l$
```

The p-multigrid algorithm is defined recursively. Although $l > p_{\min}$, the state vector $\mathbf{u}_l$ is smoothed m-times, and then, the state vector and the defect between the various approximation levels is restricted to the space $\mathbf{V}_{h,l-1}$. After these steps, the recursion takes place with adapted values. When $\tau = 1$, a V-cycle is done, whereas $\tau = 2$ produces a W-cycle. The error between the processed state vector from the lower levels, and the restricted state vector from the beginning is prolongated and added to the state vector $\mathbf{u}_l$ on level l. After that, a second smoothing takes place.

4.2. *Smoother and coarse level solver*

There are many possibilities to choose a smoother and coarse level solver, but in our experience, it has proven to be necessary to choose a rather good solving algorithm in the turbulent case. That is why in the following, we will consider the linearized backward-Euler pseudo-time stepping method. As a fully implicit method, it requires the full Jacobian of the residual. The actual solver for the linear system is typically chosen from the class of preconditioned iterative Krylov subspace methods; because of the non-symmetric Jacobian (flexible), GMRes is a common choice. Because the DG approach achieves high order of accuracy via the introduction of additional DOFs, it is not possible to use a lower-order discretization to determine an approximation to the Jacobian as this would imply treating DOFs corresponding to higher-order modes explicitly. The backward-Euler method is the standard solver of the DLR `PADGE` [20] code to solve (1):

Algorithm 2 Backward-Euler $BWE(\mathbf{u}_{l,i-1})$.

```
1: Solve $[(\alpha_i \Delta t)^{-1}\underline{\mathrm{M}} + \underline{\mathrm{R}}_l](\mathbf{u}_{l,i} - \mathbf{u}_{l,i-1}) = [\mathbf{f}_l - \mathbf{L}_l(\mathbf{u}_{l,i-1})]$,
2: return $\mathbf{u}_{l,i}$
```

where $\underline{\mathrm{R}}_l$ is the fully implicit Jacobian matrix and $\underline{\mathrm{M}}$ is the mass matrix. In addition to that, $\mathbf{u}_{l,j}$ is a given state vector, with $\mathbf{u}_{l,j} \in \mathbf{V}_{h,l} \ \forall \ j \in \mathbb{N}$. This method is controlled by a pseudo-time stepping scheme $(\alpha_i \Delta t)$, based on local time steps computed from the local state and a given CFL number. The pseudo-time step acts as a stabilizing mechanism and for CFL$\rightarrow \infty$ the Newton algorithm is recovered. Smaller time steps and accordingly CFL numbers are required in the initial phase of the solution process when the current iterate solution approximation is still too far from the converged solution. A *switched evolution relaxation* (SER) [21] technique is employed to modify the CFL number during the solution process, enabling to recover the optimal behavior of the Newton algorithm in the final phase of the solution process.

Although it usually pays off to assemble and store the full Jacobian with respect to the required overall run time of the solver, this does constitute a considerable memory requirement. Fortunately, the GMRes algorithm requires only matrix–vector products, which can be implemented in a matrix-free fashion using finite differences of the residual, either a central approximation,

$$\left[(\alpha_i \Delta t)^{-1}\underline{\mathrm{M}} + \underline{\mathrm{R}}_l\right]\mathbf{x} \approx \frac{\mathbf{L}_l(\mathbf{u}_l + \varepsilon\mathbf{x}) - \mathbf{L}_l(\mathbf{u}_l - \varepsilon\mathbf{x})}{2\varepsilon} + (\alpha_i \Delta t)^{-1}\underline{\mathrm{M}}\mathbf{x}, \quad (8)$$

which is second-order accurate, or a simple one-sided first-order accurate approximation,

$$\left[(\alpha_i \Delta t)^{-1}\underline{\mathrm{M}} + \underline{\mathrm{R}}_l\right]\mathbf{x} \approx \frac{\mathbf{L}_l(\mathbf{u}_l + \varepsilon\mathbf{x}) - \mathbf{L}_l(\mathbf{u}_l)}{\varepsilon} + (\alpha_i \Delta t)^{-1}\underline{\mathrm{M}}\mathbf{x}, \quad (9)$$

which has the advantage that $\mathbf{L}_l(\mathbf{u}_l)$ is constant during one linear solution process and thus does not have to be recomputed. However, in particular for turbulent flow, we have observed several cases in which the additional accuracy of (8) was required for stability reasons. The differencing parameter has to be chosen large enough to avoid cancelation errors and small enough to provide an accurate approximation of the derivative. Considering the relative scales of $\mathbf{u}_l$ and $\mathbf{x}$, a reasonable choice is given by $\varepsilon = \varepsilon_0 \frac{\|\mathbf{u}_l\|}{\|\mathbf{x}\|}$ with some constant ε_0 that depends on the floating point accuracy, $\varepsilon_0 = 10^{-6}$ in our examples.

Although the storage of the system matrix can be avoided with this matrix-free implementation technique, a good preconditioner is still required to allow for reasonable linear convergence rates in the Krylov process. For matrix-based implementations, typically incomplete LU factorizations provide a good compromise between memory requirement and performance, in particular the version with no additional fill-in, ILU(0), is often proposed. It requires the same storage as the original system matrix. For a matrix-free implementation, this preconditioner is the memory bottleneck, thus it is tempting to use a different approach.

Here, we consider a semi-implicit method, which keeps the important couplings in the system matrix while dropping others. In particular, we consider lines of elements, that is, disjoint subsets of elements. Each line is contiguous and contains no circles, and each element has at most two neighbors in the same line. Selecting the system matrix blocks for elements on the diagonal as well as interface blocks corresponding to couplings over a common interface between neighboring elements within a line yields a tri-diagonal block system per line, which gives the opportunity to invert the underlying problem more efficiently, for example, with the Thomas algorithm [22]. Furthermore, the inversion can be applied exactly, whereas ILU represents an approximate inverse, only. However, the underlying matrix to be inverted is only an approximation of the fully implicit matrix. Thus, in both cases the inverse of the Jacobian is in-exact w. r. t. the original matrix, and it is not a priori clear which approach yields better results.

To further improve upon this behavior, we can use a Jacobi-type iteration on the full matrix, where the inverted part of the matrix—typically the diagonal—is chosen as the subset of the full matrix contained in lines. Let $\underline{A} = (\alpha_i \Delta t)^{-1}\underline{M} + \underline{R}_l$ denote the full system matrix, $\underline{L}$ the sub-matrix extracted along lines and $\underline{R} = \underline{A} - \underline{L}$ the remaining matrix blocks, then an iterative Jacobi-type scheme for the linear system $\underline{A}\mathbf{x} = \mathbf{b}$ is given by

$$\mathbf{x}^{(k+1)} = \underline{L}^{-1}[\mathbf{b} - \underline{R}\mathbf{x}^{(k)}]. \tag{10}$$

Subtracting the last iterate $\mathbf{x}^{(k)}$ on both sides of (10) and introducing $\Delta\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$, the process can be written in terms of $\underline{A}$ instead of $\underline{R}$, eliminating the need to form the latter explicitly:

$$\Delta\mathbf{x}^{(k+1)} = \underline{L}^{-1}[\mathbf{b} - \underline{A}\mathbf{x}^{(k)}], \qquad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k+1)}. \tag{11}$$

Again, it is not necessary to store and compute the whole Jacobian, as the required matrix–vector product can be evaluated in a matrix-free fashion via finite differences. Using this approach, an iterative solver is used as the preconditioner, where intra-line couplings are respected exactly via the inversion along lines, whereas inter-line couplings are treated in a weaker form via multiplication with the full system matrix to form the new linear residual. Typically, only very few iterations are taken. The outer Krylov iteration takes care of all error modes that cannot efficiently be reduced using the line-Jacobi iteration.

In order to find suitable lines, a scalar advection–diffusion problem is defined and examined [7]. The lines, which are created in the diffusion, dominated part of the flow field align in the main direction of diffusive effects, for example, in boundary layers near solid walls, the lines are orthogonal to those walls, thus resolving mesh anisotropies. In the convection dominated part, for example, the farfield, the lines follow the convective flow direction. This ansatz tries to ensure that those couplings in the Jacobian, which are most relevant for solving the DG discretization of (1), are retained in the lines. On the other hand, we do not compute the weaker couplings in the Jacobian.

There exist other algorithms to find lines in the mesh, some of which follow a geometrical approach [8]. This algorithm will create lines at anisotropies in the mesh using the inverse distance between cell mid points, which is very similar to the effect of a scaled diffusion operator. Isotropic regions will be treated using trivial lines, each with a single element.

The third possibility is to use only trivial lines. This means that every cell in the mesh is a separate line. In this way, element-implicit and line-implicit methods are treated in a unified framework. Although the latter approach simplifies the algorithm and its implementation, it is not suited to reduce mesh anisotropies.

In general, lines defined via the advection–diffusion approach are solution-dependent and could be re-computed along with the original RANS problem for maximal effectivity. However, this complicates the code infrastructure, in particular if computations are to be done in parallel and lines should not cross domain boundaries. As a compromise, lines are only determined at the beginning of each computation on a new level, that is, taking into account the flow field evaluated for a lower order discretization. The overall process is initiated with free stream velocity in the whole computational domain.

The Jacobian on trivial lines consist of a single block entry per element, whereas an inner element of a longer line contributes three blocks: one for inter-element couplings and one additional block for both the preceding and subsequent element in the same line. Thus, depending on the number and length of lines, the memory consumption for storing the Jacobian terms of the line-based matrix-free solver is between one and three blocks per element in the mesh. In contrast to that, the matrix-based full Jacobian solver requires $2(1 + N_{\text{nb}})$ blocks per element neglecting mesh boundaries. Here, N_{nb} is the number of neighbors per element, $N_{\text{nb}} = 4$ for quadrilaterals and $N_{\text{nb}} = 6$ for hexahedra. The factor of 2 arises from the need to store both the original matrix and the ILU(0) preconditioner. Thus, memory savings between 70% and 90% can be expected for structured meshes in 2D. In 3D, the savings increase to values between 79% and 93%.

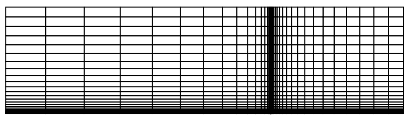
5. NUMERICAL RESULTS

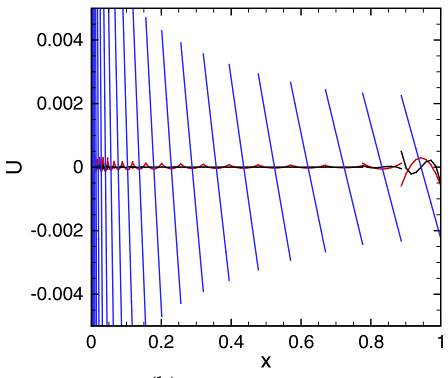
The performance of the flux functions as well as the multigrid solver will now be demonstrated regarding a range of numerical examples for aerodynamic flows. All computations have been performed using the DG flow solver `PADGE` [20], which is based on a modified version of the `deal.II` finite element library [23].

5.1. Turbulent flat plate

As a first test case, we consider the turbulent boundary layer development at a flat plate at free-stream Mach number $M_\infty = 0.3$ and Reynolds number $Re = 10 \times 10^6$. The computational mesh is rather coarse and consists of only 2240 quadrilateral elements. The mesh size is graded to better resolve the boundary layer and the leading edge of the plate, see Figure 1(a). Although the mesh is structured, the solver works on unstructured meshes and makes no use of the underlying structure of the mesh.

5.1.1. Accuracy of results for different choices of the numerical flux. First, we concentrate on the effect of the numerical flux function on the computed solution. To this end, we evaluate the normalized tangential velocity $U = \frac{v_1}{M_\infty a_\infty}$ directly at the surface of the plate. At viscous walls, this velocity should vanish, but in the DG context boundary conditions are treated in a weak form,


(a) computational mesh



(b) tangential wall velocity

Figure 1. Turbulent flat plate: (a) computational mesh and (b) tangential wall velocity using the local Lax–Friedrichs flux for second (blue), third (red) and fourth order (black) computations.

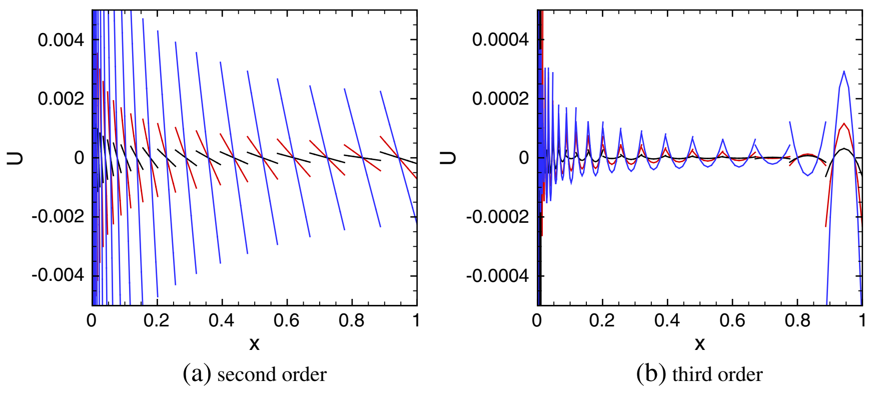


Figure 2. Turbulent flat plate: Tangential wall velocity using the local Lax–Friedrichs (blue), Roe (red) and Vijayasundaram fluxes (black) for second and third-order computations.

so the solution actually shows a non-vanishing velocity at the wall. This small deviation from the theoretical values is an indication of the solution quality.[‡]

Figure 1(b) shows the tangential wall velocity for computations using the local Lax–Friedrichs flux for polynomial degrees ranging from 1 to 3, that is, for methods that are designed for second to fourth-order accuracy. In all cases, the average value is very close to zero, but there are spurious, local, element-wise oscillations. These deviations are larger towards the leading edge singularity and very close to the outflow boundary at $x = 1$, at which the imposed farfield boundary condition is not really appropriate for the outflow of a viscous boundary layer. Apart from the latter point the size of the maximal deviation decreases with increasing order of the method, that is, the numerical results converge against the exact values with increasing resolution of the method, but on any given mesh and order, the errors have a certain size.

Figure 2 illustrates, that the absolute size of the considered error on a given mesh and at a given order of the method can be influenced strongly by the choice of the numerical flux function. Although all flux functions show a similar behavior in qualitative terms, the local extrema are significantly reduced when the Roe flux is used instead of the local Lax–Friedrichs flux. Obviously, approximating all eigenvalues with the largest one yields a stable but not very accurate method. The size of the deviations can further be reduced using the Vijayasundaram flux. For both second and third-order methods, the maximal deviation is reduced by an order of magnitude compared with the local Lax–Friedrichs flux. The scale of the U-axis is reduced for the third order plot.

Several authors stress the fact that a high-order DG method can be achieved irrespective of the flux and thus advocate the use of the local Lax–Friedrichs flux due to its simplicity and robustness. However, although it is true that the order of convergence is not affected, the error constant might strongly be affected, thus the additional complexity of the Roe flux might well be worth the effort. This corresponds to similar findings for the comparison of scalar versus matrix dissipation in central schemes for finite volume discretizations.

5.1.2. Preconditioner behavior. As a second aspect, we will now examine the behavior of the proposed preconditioner with regard to parameter choices as well as in comparison with an ILU(0) preconditioner. In order to minimize discarded fill-in, the DOFs are reordered based on the reverse Cuthill–McKee algorithm [24]. Typically, for parallel applications based on domain decomposition approaches, ILU preconditioners are not really implemented in a parallel way due to the required communication overhead. Instead, they are in general applied only per sub-domain, on the global problem only a single Jacobi-type iteration based on the sub-problems is performed. This yields a preconditioner with decreasing performance on an increasing number of sub-domains. In contrast

[‡]In addition to the discretization error we are interested here, the presented plots contain an error due to the use of an iterative solver, which does not yield an exact solution of the discrete equations. However, this effect is rather small. Furthermore, all results are shown for the same level of iterative convergence of the solver.

Table I. Comparison of preconditioner choices for the flat plate at $p = 2$. The number in parentheses for geometric lines indicates the threshold anisotropy to connect two elements, larger numbers indicate a stricter requirement and thus shorter lines. For the geometric lines denoted with *, a two phase strategy is applied: after the initial line identification, the lines are additionally connected to create longer lines, in particular this attaches single elements in isotropic parts of the mesh to neighboring lines.

Preconditioner	Jacobi it.	Non-linear it.	Total linear it.	Equiv. linear it.	CPU time
ILU(0)	1	27	1309	1309	1
Adv.–diff. lines	3	40	216	648	1.65
Adv.–diff. lines	4	26	147	588	1.13
Adv.–diff. lines	5	27	121	605	1.18
Geom. lines (2)	1	28	578	578	1.18
Geom. lines (2)	2	27	308	616	1.15
Geom. lines (2)	3	27	211	633	1.15
Geom. lines (2)	4	27	168	672	1.17
Geom. lines (2)	5	27	132	660	1.17
Geom. lines (4)	1	28	738	738	1.22
Geom. lines (4)	3	26	255	765	1.14
Geom. lines (8)	1	28	933	933	1.26
Geom. lines (8)	4	27	236	944	1.20
Geom. lines (8)	5	27	196	980	1.21
Geom. lines (2)*	1	31	510	510	1.37
Geom. lines (2)*	4	27	141	564	1.22
Geom. lines (2)*	5	27	116	580	1.22

to that, the partitioning in our code respects the lines and makes sure, that no line is cut by domain boundaries. Thus, the algorithmic performance of the line-Jacobi preconditioner is independent of the number of sub-domains. To include this effect in the comparison, we compute the third- order case with the Roe flux from the example earlier using four parallel sub-domains. Furthermore, to separate the preconditioner from other effects, it is applied within a matrix-based implementation of the Krylov method using a single level algorithm. Therefore, only the preconditioner is varied compared with the current standard solver algorithm within the PADGE code. The Krylov process for each linear problem is stopped when the linear residual is reduced by one order of magnitude or else after 120 iterations. The non-linear iteration is stopped after the non-linear residual is reduced below 10^{-10}. The initial CFL number is set to 5 and the computation is started from a converged second-order solution.

Table I summarizes the performance of different preconditioner choices. As expected, the number of nonlinear iterations does not vary much between the choices, as the preconditioner mainly affects the performance of the linear solution process. The total number of linear iteration steps (over all non-linear cycles) can be drastically reduced using line-Jacobi preconditioning. Additionally, an equivalent number of linear iteration steps is given as the product of linear steps and the number of Jacobi iterations in the preconditioner, as this is a better indicator for the real effort: this product counts the number of inverted matrix and system matrix applications. Still, also this number can be reduced with the line-Jacobi preconditioner, indicating that ILU(0) is a rather simple choice, but not necessarily the most effective one.

For the geometric lines, it is beneficial to use a low threshold for the mesh anisotropy in order to create longer lines, see Figure 3, which illustrates the different lines used in the computations. All approaches correctly identify lines in strongly stretched mesh regions. Extending the lines into the isotropic part of the mesh can be helpful up to a certain point. Additionally, attaching individual elements to the created lines as in the * version in Table I can improve this behavior even more, but it might decrease the robustness: Preconditioners with two or three Jacobi iterations did not converge for this particular setting, thus increasing the length of lines can also be detrimental. The reason for this might be found in the rather strange and non-intuitive lines used in this case,

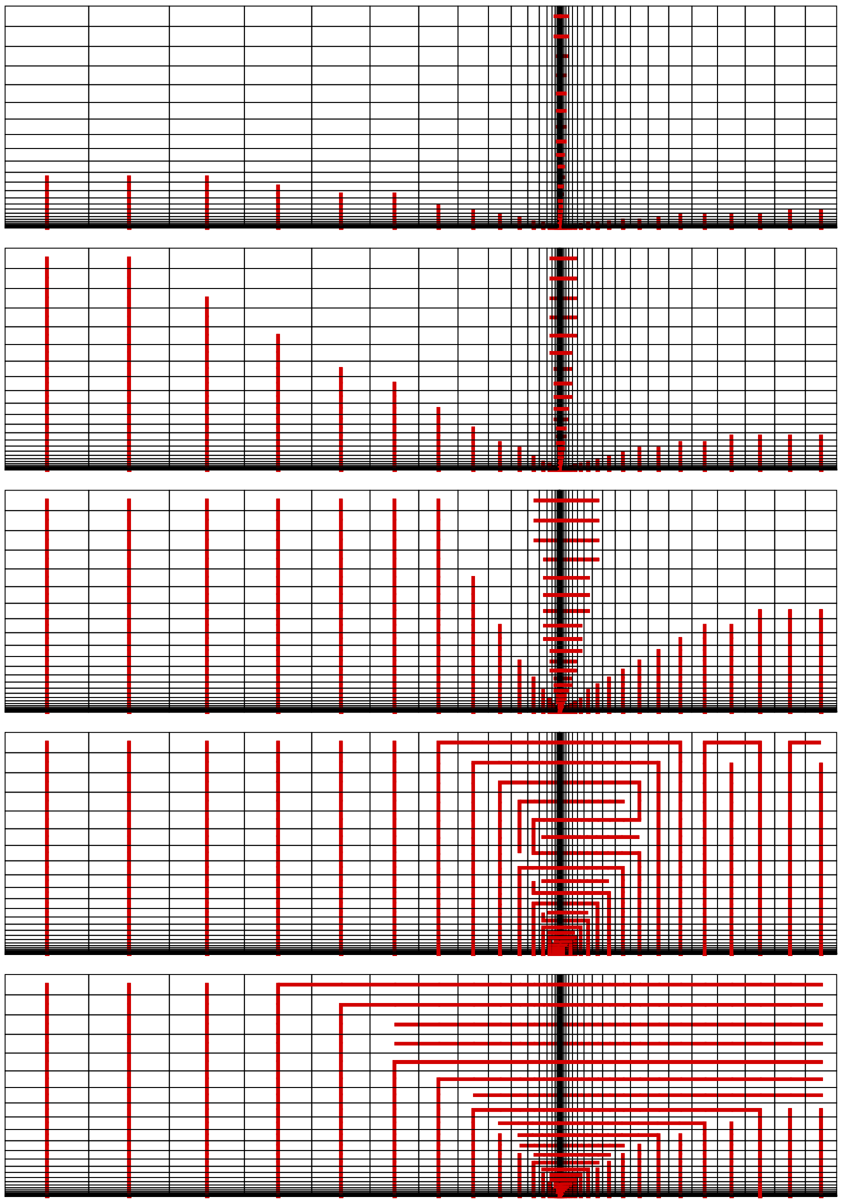


Figure 3. Lines computed for the turbulent flat plate. From top to bottom: (a) to (c) geometric lines with an anisotropy threshold of 8, 4 and 2; (d) geometric lines with an anisotropy threshold of 2 and additional line connection; and (e) advection–diffusion lines.

see Figure 3. In contrast to that, lines based on an advection–diffusion equation switch from resolving strong anisotropies to following the convection direction in more isotropic parts of the mesh, which is better suited to reflect the strong couplings in the underlying physics and numerics, thus, they seem to be a better and in particular more robust choice, in general. For the case at hand, trivial single-element lines could not be used successfully, as the non-linear convergence stagnates before the residual is reduced below 10^{-6}.

For both geometric lines as well as lines determined on the basis of an advection–diffusion problem, increasing the number of Jacobi iterations can be beneficial, but after a certain number, the additional gain in the linear iterations is not enough to counteract the additional effort per step. An optimum is found for a relatively small number of Jacobi iterations. In general, well tuned geometric lines can be as efficient as advection–diffusion lines, but there is a large variation, and results for other cases not shown in detail here indicate that, in general, the auxiliary advection–diffusion problem yields good behavior of the line-Jacobi preconditioner with increased robustness.

The CPU time included in Table I has been normalized with the run time of the baseline ILU(0) preconditioned case. Interestingly, the baseline case is the fastest in spite of the fact that the algorithmic effort has been reduced considerably for other preconditioner choices. This is probably due to a sub-optimal implementation in the current code, in particular the system matrix is assembled twice, once for the Krylov iteration and once for the lines. This is due to the fact that no optimization has been performed for the matrix-based case because the matrix-free implementation is the goal of the current efforts. Furthermore, the ILU case is realized using the rather efficient implementation in PETSc [25, 26]. Correcting the timings for this effect, the run time of the line-Jacobi preconditioner can actually be decreased well below the baseline, but even so, the performance is at least not considerably reduced, whereas the memory consumption is in fact reduced. Thus, the aims of this algorithm are achieved. The timings between the different line-preconditioned choices do not correspond exactly to the equivalent number of linear iterations. This is due to different effects, among them the fact that the number of non-linear iterations and thus assemble procedures of the Jacobian matrix has a large influence on the run time as well as the fact that the complexity of a Krylov process does not depend linearly on the number of iterations.

Finally, the matrix-free version of the line-Jacobi preconditioned algorithm shows almost the same algorithmic performance as the matrix-based version. For the preconditioner based on advection–diffusion lines and four Jacobi iterations, it converges in 27 non-linear iterations and 572 equivalent linear steps. The reduced memory requirements—no system matrix needs to be stored for this variant—is paid for by an increase in normalized run time to 2.47, which yields an increase by a factor of approximately 2.2 compared with the matrix-based algorithm in this case and for our implementation.

5.1.3. p-multigrid behavior. Finally, the convergence behavior of the multigrid algorithm is analyzed. We consider again the $p = 2$ case, this time using the local Lax–Friedrichs flux. We choose a preconditioner with four Jacobi iterations based on advection–diffusion lines. Only the matrix-free version of the algorithm is used. The linear solution process is stopped when the linear residual is reduced by at least a factor of 0.3 or after 30 Krylov iterations. These weak tolerances are intended to use the linear iteration as a smoother only, not as a solver. The computation is again started from a converged $p = 1$ solution. The performance of this algorithm is compared for a single-level, two level ($p = 1, 2$) and three level ($p = 0, 1, 2$) V-cycle multigrid algorithm in Table II.

Using two instead of just one level, the number of required non-linear cycles is reduced. It has to be noted that the multigrid algorithm employs one pre-smoothing and one post-smoothing step on the finest level per cycle, thus the total number of linear problems is actually increased in the two-level algorithm. However, in particular, the post-smoothing iteration converges extremely fast, it is basically just a correction for the imperfect prolongation operator, thus, the overall number of linear iterations can be reduced because of the additional work on the coarse level. Using three levels in the multigrid, the initial CFL number can be increased to five, which further reduces both the non-linear cycles as well as the number of fine level linear iterations. However, the overall savings are so small that the total run time is increased because of the overhead in the multigrid case compared with the single-grid case, see also the residual reduction over time in the left part of Figure 4. This might be due to the fact that the mesh is quite coarse and that the smoother is a

Table II. Comparison of single-grid and multigrid performance for the flat plate at $p = 2$.

Initial CFL	Levels	Non-linear cycles	Linear it. on fine level	CPU time
3.5	1	34	371	1
3.5	2	25	334	1.17
5	3	21	291	1.19

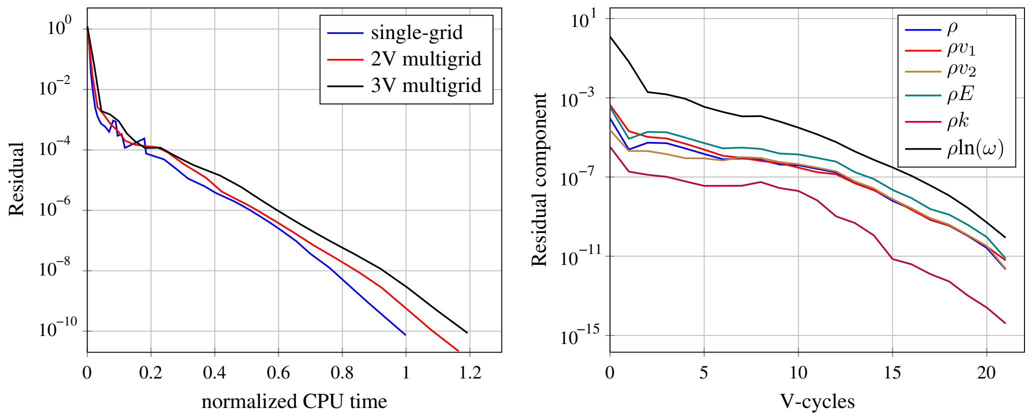


Figure 4. Total residual versus run time for single-grid and multigrid flat plate computation at $p = 2$ (left) as well as residual components for the multigrid (3V) case (right).

rather good solver itself. However, using a weaker solver, the solution cannot be converged at all or only at a reduced overall efficiency. This result is somewhat in contrast to findings of other authors [4–7], but the differences might be explained by the fact that we consider stronger smoothers and more difficult stiff problems arising from high Reynolds numbers, turbulence models and strongly stretched meshes—the maximum aspect ratio in the flat plate mesh is as large as 120,000.

The plot of individual residual components in the right part of Figure 4 shows a similar behavior for all equations. After an initial reduction, the iterative convergence rate is rather low, in particular for the turbulent kinetic energy (ρk) equation, but then the convergence rates increase. The use of an adaptive CFL number enables the recovery of Newton-like convergence rates, but because of the inexact linear solution process, the asymptotic iterative convergence rate is bounded. Still, the residuals are reduced by more than half an order of magnitude per cycle in the final phase of the non-linear solution process, indicating the overall satisfying performance of the approach.

Several authors have looked at the scaling properties of p-multigrid algorithms. Concerning the scaling with p, the results are good in the current case: The $p = 1$ solution using a two-level algorithm takes 72 non-linear cycles with a total of 581 linear iterations on the fine level. Both numbers are considerably larger than for the $p = 2$ case. This might be explained by the following facts: (i) that the number of levels is reduced for $p = 1$; (ii) that the $p = 0$ level is a rather bad approximation of higher order representations (it even yields an inconsistent discretization for source terms); and (iii) that the initial solution is not as good as in the $p = 2$ case.

However, scaling with p is only of limited interest in cases where mesh resolution instead of high order might be used more efficiently, for example, for non-smooth flows or to resolve complex geometries. In order to estimate the scaling properties with the mesh size, the same case is repeated on coarsened mesh. Because of the underlying structure of the mesh, four elements can be fused into one, and the resulting mesh is coarser by a factor of exactly four. On this coarser mesh, 20 non-linear 3V-cycles are required for the $p = 2$ solution, which is very similar to the 21 cycles on the finer mesh. However, the number fine level linear iterations is reduced to 187, that is, less than 65% of the fine mesh value. Correspondingly, the total run time on the coarse mesh is reduced by a factor of 4.8 instead of just 4.0, indicating the expected result that the p-multigrid algorithm does not scale linearly w.r.t. the mesh size because of the fact that the coarse level problem size cannot be reduced enough.

5.2. *RAE 2822 airfoil*

As a second test case, we examine the turbulent flow around the RAE 2822 airfoil at a Reynolds number $Re = 1 \cdot 10^6$, a Mach number $M_\infty = 0.4$ and an angle of attack of $\alpha = 1°$. A structured quadrilateral mesh with 2024 elements is used. In order to better represent the curved geometry, the mesh edges are treated as quartic polynomials. Because of the strong stretching of boundary layer

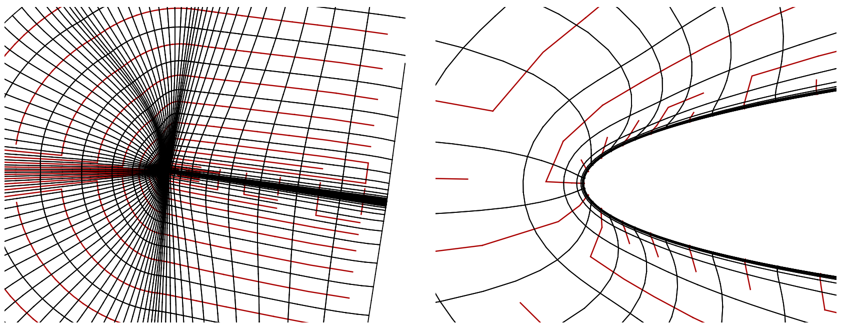


Figure 5. Computed lines in the mesh for the RAE 2822.

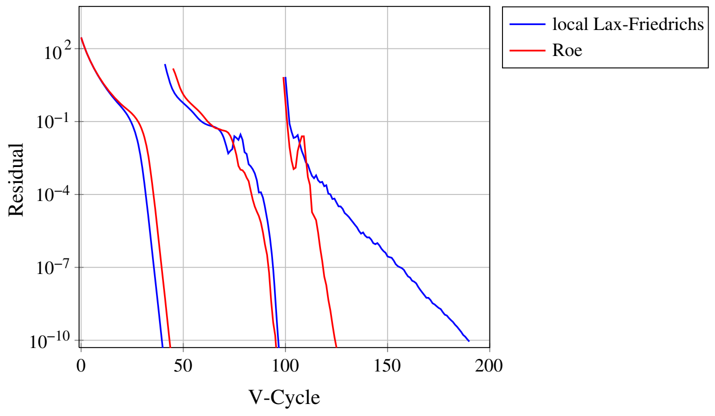


Figure 6. RAE 2822 airfoil: residual plot of two full p-multigrid computations from $p = 0$ to 2.

elements, it is necessary to treat (at least a part of) the interior elements as curvilinear. For simplicity, all elements are curved in this case. We compute lines with the advection–diffusion problem for the turbulent flow around the RAE 2822 airfoil, see Figure 5.

Figure 6 shows the convergence behavior of two full p-multigrid (FMG) computations for $p = 0$ up to 2, with $p_{\min} = p - 1$. At every level $l > p_{\min}$, one pre-smoothing and post smoothing step is done with the backward-Euler solver and line-Jacobi preconditioner introduced in 4.2. Moreover on $p_{\min}$, one backward-Euler step is done as coarse level solver, again using the line-Jacobi preconditioner. The only difference between these two FMG computations is the numerical flux, all other solver adjustments are the same.

For these solver adjustments, we see a faster iterative convergence when we use the Roe flux instead of the local Lax–Friedrichs flux in the p-multigrid. Enhancing the discretization seems to be a good approach while using the p-multigrid algorithm. Again, the main idea is to use the backward-Euler algorithm more as a smoother instead of a solver. Therefore, on every level, a single backward-Euler step is performed. Furthermore, within each of these steps, the underlying linear problem is solved only approximately until the linear residual is reduced by one order of magnitude. Figure 7 shows that with these weak solver settings, a single-grid (SG) computation fails, for both Roe and local Lax–Friedrichs flux. The p-multigrid has a stabilizing effect in this case and allows the non-linear solution process to converge with this weak solver settings within the Krylov solver. As it can be very difficult to obtain a good linear solution for stiff problems, this feature might be more important for applications to complex problems.

5.3. L1T2 three element airfoil

As a last test case, we apply the multigrid algorithm to a two-dimensional high lift case. To this end, we consider the turbulent flow around the L1T2 airfoil [27] at a Reynolds number $Re = 3.52 \times 10^6$,

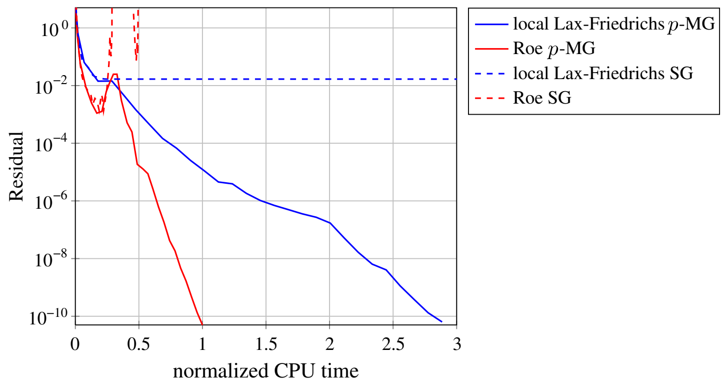


Figure 7. RAE 2822 airfoil: residual plot for single-grid and p-multigrid solvers at $p = 2$.

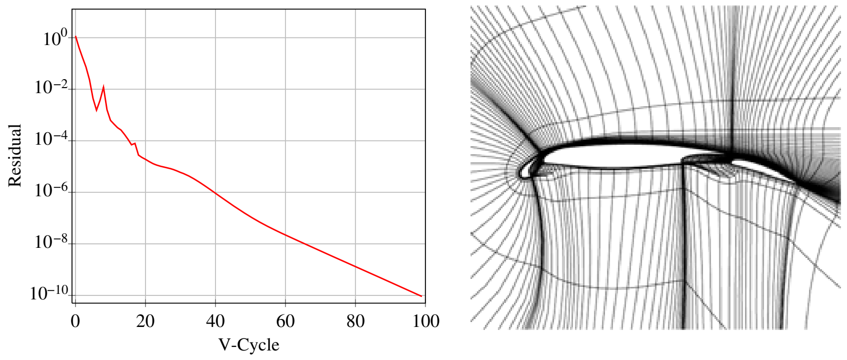


Figure 8. L1T2 three element airfoil: p-multigrid residual plot for $p = 3$ (left) and close-up view of the structured mesh (right).

a Mach number $M_\infty = 0.197$ and an angle of attack of $\alpha = 20.18°$. It has been previously demonstrated that the DG approach is well suited to tackle this case [2]. The computational mesh consists of only 4740 curvilinear quadrilateral elements and is obtained from a finer structured mesh via agglomeration. In this test case, characteristic farfield boundary condition and the Roe flux are applied, see Section 3. Figure 8 shows a p-multigrid computation for $p = 3$ with $p_{\min} = 2$. As coarse level solver and smoother, the backward-Euler with matrix computation is used. The line-Jacobi preconditioner, introduced in 4.2, with three preconditioner steps and advection–diffusion lines is applied.

A single grid computation with the same preconditioner and discretization is faster by a factor of 1.6. In this test case, a good smoother is required which also acts as a good solver, thus all relevant error frequencies can also be reduced on the fine level alone. In this situation, no additional gain can be taken from the p-multigrid. Still, this case demonstrates the ability of the algorithm to also cope with such a complex flow.

6. CONCLUSION AND OUTLOOK

We have derived a consistent linearization of the convective fluxes encountered in the completely coupled RANS+$k\omega$ equations. Because of the relatively weak couplings in the convective terms, the results are rather similar to the well-known linearization of the Euler equations. This linearization can be used to derive the simple local Lax–Friedrichs flux or the Roe and Vijayasundaram fluxes. The latter has been shown to correspond very well to the Roe flux without entropy fix, which explains both its accuracy and its instability. Numerical examples verify that the accuracy of DG RANS computations depends strongly on the choice of the numerical flux function. Whereas prior results [2] already showed that it is possible to achieve high order of convergence using the simple local Lax–Friedrichs flux, the error constant and thus the absolute quality of the computation on a given mesh at given order can be reduced when using more accurate formulations such as the Roe flux. Recent results outside the scope of this work indicate that the choice of an appropriate flux function is also of high importance in the case of transonic flows with shocks.[§]

Furthermore, the numerical flux induces an oscillating error mode that is associated with the resolution achieved with both the mesh and the order of the method. Thus, reducing this error mode with a more accurate flux formulation can actually lead to an improved behavior of a multigrid algorithm. We believe that this is due to the fact that the coarse level problem is a better approximation of the fine level problem if an appropriate linearization is used in the formulation of the numerical flux. This assumption was the underlying idea of the current work and has been verified numerically.

We also presented a framework for p-multigrid based on a backward-Euler smoother that is solved with a preconditioned Krylov method. We considered matrix-free implementations of the basic Krylov solver and memory-lean alternatives to ILU preconditioners based on line-Jacobi iterations. Both components have been demonstrated and can be used efficiently, independent of a multigrid formulation. This new framework naturally includes the current state-of-the-art single-grid ILU-preconditioned matrix-based backward-Euler solver of the `PADGE` [20] code and thus allows for a rigorous comparison. It also enables to recover other proposed algorithms. If the matrix-free formulation is used with one-sided differences using $\varepsilon = 1$ and only the preconditioner is applied without any real Krylov solver, the recovered algorithm is very similar to the semi-implicit Runge–Kutta formulation of Bassi *et al.* [5], only extended to more general lines instead of just elements.

The use of a rather strong smoother, which could also act as solver is guided by our experience that it is very hard to find a smoother that yields any iterative non-linear convergence at all in the context of turbulent flow. However, it also allows to analyze the ultimate effectivity of p-multigrid. In the limit, if the smoother on the fine level acts as a solver, eliminating all relevant error modes, no additional gain can be expected from the multigrid formulation. However, we have seen several cases in which the use of multigrid is detrimental. Part of this effect might be due to non-optimal implementations and a corresponding overhead of the resulting algorithm. However, other results not reported here indicate that also the algorithmic performance can be worse. In some cases, the p-multigrid stagnates or even diverges in cases where the single-grid solver converges with the same linear solver settings. Although we still search for possible remedies to this situation, it is our belief that the very simple underlying idea of the p-multigrid might not be as accurate as it seems, that is, a low order DG discretization might not always be a good, appropriate approximation of the high-order discretization on the same mesh, in particular for stiff complex problems. Thus, further research will be devoted to additionally investigate the use of an agglomeration-based h-multigrid directly for the high-order DG discretization. Such an approach would include the use of basis functions defined in the real space (not on reference elements) and the integration over the fine mesh elements in order to obtain high-order Gaussian quadrature rules [28].

APPENDIX A: CONVECTIVE FLUX JACOBIAN

The Jacobian of the normal convective flux in 3D is defined as follows:

$$\mathcal{F}_n^{c\,\prime} := \frac{\mathrm{d}(\mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n})}{\mathrm{d}\mathbf{u}} = \sum_{j=1}^{3} n_j \cdot F_j, \quad j = 1, 2, 3, \quad \text{with}$$

[§]The presence of discontinuities (shocks) in the solution would also require a dedicated treatment within a multigrid algorithm.

$$F_1 =$$
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -v_1^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (3-\gamma)v_1 & (1-\gamma)v_2 & (1-\gamma)v_3 & \gamma-1 & \frac{5}{3}-\gamma & 0 \\ -v_1 v_2 & v_2 & v_1 & 0 & 0 & 0 & 0 \\ -v_1 v_3 & v_3 & 0 & v_1 & 0 & 0 & 0 \\ (\gamma-1)v_1\mathbf{v}^2 - \gamma v_1 E + (\gamma - \frac{5}{3})v_1 k & \gamma E - \frac{\gamma-1}{2}(2v_1^2 + \mathbf{v}^2) + (\frac{5}{3}-\gamma)k & (1-\gamma)v_1 v_2 & (1-\gamma)v_1 v_3 & \gamma v_1 & \left(\frac{5}{3}-\gamma\right)v_1 & 0 \\ -v_1 k & k & 0 & 0 & 0 & v_1 & 0 \\ -v_1\omega & \omega & 0 & 0 & 0 & 0 & v_1 \end{pmatrix},$$

$$F_2 =$$
$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -v_1 v_2 & v_2 & v_1 & 0 & 0 & 0 & 0 \\ -v_2^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (1-\gamma)v_1 & (3-\gamma)v_2 & (1-\gamma)v_3 & \gamma-1 & \frac{5}{3}-\gamma & 0 \\ -v_2 v_3 & 0 & v_3 & v_2 & 0 & 0 & 0 \\ (\gamma-1)v_2\mathbf{v}^2 - \gamma v_2 E + \left(\gamma - \frac{5}{3}\right)v_2 k & (1-\gamma)v_1 v_2 & \gamma E - \frac{\gamma-1}{2}(2v_2^2 + \mathbf{v}^2) + \left(\frac{5}{3}-\gamma\right)k & (1-\gamma)v_2 v_3 & \gamma v_2 & \left(\frac{5}{3}-\gamma\right)v_2 & 0 \\ -v_2 k & 0 & k & 0 & 0 & v_2 & 0 \\ -v_2\omega & 0 & \omega & 0 & 0 & 0 & v_2 \end{pmatrix},$$

$$F_3 =$$
$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -v_1 v_3 & v_3 & 0 & v_1 & 0 & 0 & 0 \\ -v_2 v_3 & 0 & v_3 & v_2 & 0 & 0 & 0 \\ -v_3^2 + \frac{\gamma-1}{2}\mathbf{v}^2 & (1-\gamma)v_1 & (1-\gamma)v_2 & (3-\gamma)v_3 & \gamma-1 & \frac{5}{3}-\gamma & 0 \\ (\gamma-1)v_3\mathbf{v}^2 - \gamma v_3 E + \left(\gamma - \frac{5}{3}\right)v_3 k & (1-\gamma)v_1 v_3 & (1-\gamma)v_2 v_3 & \gamma E - \frac{\gamma-1}{2}(2v_3^2 + \mathbf{v}^2) + \left(\frac{5}{3}-\gamma\right)k & \gamma v_3 & \left(\frac{5}{3}-\gamma\right)v_3 & 0 \\ -v_3 k & 0 & 0 & k & 0 & v_3 & 0 \\ -v_3\omega & 0 & 0 & \omega & 0 & 0 & v_3 \end{pmatrix}.$$

ACKNOWLEDGEMENTS

The authors would like to thank Florian Prill for his work on solver algorithms in the PADGE code, in particular, on the implementation of a *p*-multigrid strategy for inviscid and laminar flows [29]. Furthermore, we also express our gratitude for many fruitful discussions with Bernhard Eisfeld and Stefan Langer on turbulence models and implicit solvers, as well as Ralf Hartmann on DG methods in general.

Research work within the flow solver code is supported by the European Commission funded IDIHOM project.

REFERENCES

1. Bassi F, Crivellini A, Rebay S, Savini M. Discontinuous Galerkin solution of the Reynolds-averaged Navier–Stokes and k-ω turbulence model equations. *Computers & Fluids* 2005; **34**(4-5):507–540.
2. Hartmann R, Held J, Leicht T. Adjoint-based error estimation and adaptive mesh refinement for the RANS and k-ω turbulence model equations. *Journal of Computational Physics* 2011; **230**(11):4268–4284.
3. Landmann B, Kessler M, Wagner S, Krämer E. A parallel, high-order discontinuous Galerkin code for laminar and turbulent flows. *Computers & Fluids* 2008; **37**(4):427–438.
4. Burgess N, Nastase C, Mavriplis D, Martinelli L. Efficient solution techniques for discontinuous Galerkin discretizations of the Navier–Stokes equations on hybrid anisotropic meshes, 2010. 48th AIAA Aerospace Sciences Meeting, AIAA 2010-1448.
5. Bassi F, Ghidoni A, Rebay S, Tesini P. High-order accurate *p*-multigrid discontinuous Galerkin solution of the Euler equations. *International Journal for Numerical Methods in Fluids* 2009; **60**:847–865.
6. Luo H, Baum JD, Löhner R. A *p*-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. *Journal of Computational Physics* 2006; **211**(2):767–783.
7. Fidkowski KJ, Oliver TA, Lu J, Darmofal DL. *p*-multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations. *Journal of Computational Physics* 2005; **207**(1): 92–113.
8. Langer S. Point and line implicit methods to improve the efficiency and robustness of the DLR TAU code. In *New Results in Numerical and Experimental Fluid Mechanics VIII: Contributions to the 17th STAB/DGLR Symposium* Berlin, Germany 2010, Kreplin HP (ed.), Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Springer. To appear.
9. Wilcox DC. Reassessment of the scale-determining equation for advanced turbulence models. *AIAA Journal* 1988; **26**(11):1299–1310.
10. Wilcox DC. *Turbulence Modeling for CFD*. DCW Industries Inc.: La Canada CA, 1993.
11. Bassi F, Rebay S, Mariotti G, Pedinotti S, Savini M. A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows. In *2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics March 5–7, 1997*, Decuypere R, Dibelius G (eds). Technologisch Instituut: Antwerpen, Belgium, 1997; 99–108.
12. Roe PL. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics* 1981; **43**(2):357–372.

13. Mavriplis D. Unstructured mesh discretizations and solvers for computational aerodynamics. *18th AIAA Computational Fluid Dynamics Conference*, Miami, Florida, 2007; 28 pages. AIAA 2007-3955.
14. Lax PD. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications on Pure and Applied Mathematics* 1954; **7**(1):159–193.
15. Rusanov VV. Calculation of the interaction of unsteady shock waves with obstacles. *Z. Vycisl. Mat. i Mat. Fiz. (Computational Mathematics and Mathematical Physics)* 1961; **1**(2):267–279. In Russian.
16. Harten A, Hyman JM. Self adjusting grid methods for one-dimensional hyperbolic conservation laws. *Journal of Computational Physics* 1983; **50**(2):235–269.
17. Vijayasundaram G. Transonic flow simulations using an upstream centered scheme of Godunov in finite elements. *Journal of Computational Physics* 1986; **63**(2):416–433.
18. Langer S. Investigation and application of point implicit Runge–Kutta methods to inviscid flow problems. *International Journal for Numerical Methods in Fluids* 2011; **69**(2):332–352.
19. Blinkov YA, Gerdt VP, Robertz D. GINV-project - Gröbner bases constructed by involutive algorithms. *Technical Report*, 2011. (Available from: http://invo.jinr.ru/ginv/index.html).
20. Hartmann R, Held J, Leicht T, Prill F. Discontinuous Galerkin methods for computational aerodynamics—3D adaptive flow simulation with the DLR PADGE code. *Aerospace Science and Technology* 2010; **14**(7):512–519.
21. Mulder WA, Leer BV. Experiments with implicit upwind methods for the Euler equations. *Journal of Computational Physics* 1985; **59**(2):232–246.
22. Thomas L. Elliptic problems in linear difference equations over a network, Watson Sci. Comput. Lab. Rept., Columbia University, New York, 1949.
23. Bangerth W, Hartmann R, Kanschat G. deal.II—A general purpose object oriented finite element library. *ACM Transactions on Mathematical Software* 2007; **33**(4):24/1–24/27.
24. Cuthill E, McKee J. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings Of The 1969 24th National Conference*. ACM Press: New York, NY, USA, 1969; 157–172.
25. Balay S, Brown J, Buschelman K, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Smith BF, Zhang H. PETSc Web page, 2011. (Available from: http://www.mcs.anl.gov/petsc).
26. Balay S, Brown J, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Smith BF, *et al.* PETSc users manual. *Technical Report ANL-95/11 - Revision 3.2*, Argonne National Laboratory, 2011.
27. Moir IRM. Measurements on a two-dimensional aerofoil with high-lift devices. *AGARD Advisory Report 303*, Advisory Group for Aerospace Research & Development, Neuilly-sur-Seine, 1994. Test case A2.
28. Bassi F, Botti L, Colombo A, Rebay S. Agglomeration based discontinuous Galerkin discretization of the Euler and Navier–Stokes equations. *Computers & Fluids* 2012; **61**:77–85.
29. Prill F. Diskontinuierliche Galerkin-Methoden und schnelle iterative Löser zur simulation kompressibler Strömungen. *Phd thesis*, TU Hamburg-Harburg, 2010. In German, available as DLR-FB 2010-17, ISSN 1434–8454.