Matrix-Free Second-Order Methods in Implicit Time Integration for Compressible Flows Using Automatic Differentiation

F. D. Bramkamp [*] B. Pollul [†] A. Rasch [‡] G. Schieffer [§]

October 24, 2008

Key Words: Matrix-free methods, Newton-Krylov, Automatic differentiation, QUADFLOW, ADIFOR

Abstract

In the numerical simulation of inviscid and viscous compressible fluid flow, implicit Newton-Krylov methods are frequently used. A crucial ingredient of Krylov subspace methods is the evaluation of the product of the Jacobian matrix of the spatial operator, e.g., fluxes, and a Krylov vector. In this article we consider a matrix-free implementation of the Jacobian-vector product within the flow solver QUADFLOW using automatic differentiation. The convergence of the nonlinear iteration using first- and second-order accurate Jacobian-vector products is compared. It turns out that a hybrid implementation employing both, first- and second-order accurate methods, significantly reduces the overall execution time of the simulation.

1 Introduction

The demand for high-capacity aircraft is increasing rapidly. In order to reduce time and cost during the development of next-generation fuel-efficient aircraft, engineers rely on numerical simulation as a design tool reducing the number of expensive wind tunnel experiments. The three-dimensional simulation of the flow field around an aircraft requires accurate and efficient numerical methods in order to resolve all relevant flow features. At RWTH Aachen University the collaborative research center SFB 401 "Modulation of Flow and Fluid-Structure Interaction at Airplane Wings" [3] is concerned with fundamental problems of high-capacity aircraft in transonic conditions, such as deformation of airplane wings and wake vortices, which may cause a hazard for following aircraft. One key project in the SFB 401 is the development of a new adaptive finite volume flow solver called QUADFLOW [10, 13, 14, 11].

[*] Mechanics Department[a]
[†] Chair for Numerical Mathematics[a]
[‡] Institute for Scientific Computing[a]
[§] Chair for Computational Analysis of Technical Systems[a]
[a] RWTH Aachen University, D-52056 Aachen, Germany. This research was supported by the German Research Foundation through the Collaborative Research Centre SFB 401

In this paper, we consider stationary flow simulations. The underlying physical model is described by the Euler- or the Navier-Stokes equations, which are solved by an implicit time integration method within a pseudo-transient continuation. For the solution of the resulting nonlinear systems of equations we rely on Newton-Krylov methods [31, 37]. In order to achieve quadratic convergence of Newton's method, exact derivative information is required. Within a second-order accurate discretization framework the explicit storage of the exact Jacobian is usually prohibitive due to memory limitations. However, Newton-Krylov methods require only the product of the Jacobian matrix with a given vector, rather than explicit access to the elements of the Jacobian. Such Jacobian-vector products can be approximated by divided differences. The divided differencing method has the conceptual disadvantage that it involves the choice of a suitable step size which is typically not known a priori. An alternative is provided by a technique called automatic differentiation, which allows the computation of derivatives without truncation error. In this work we employ automatic differentiation to obtain the matrix-free evaluation of Jacobian-vector products in an efficient and accurate fashion.

In [26] Hovland and McInnes compare the matrix-free implementations based on divided differencing and automatic differentiation in a Newton-Krylov-Schwarz framework. An alternative approach by Barth and Linton [4], also followed in [41, 42], is based on analytic derivation of the Jacobian, except for the limiter functions. Second- and higher-order matrix-free methods for unstructured meshes are compared in [40], where the Jacobian-vector product is calculated via divided differences.

In this paper, we employ a second-order accurate finite volume scheme, where the Jacobian is based either on the first- or second-order accurate discretization, yielding an approximate or exact Newton method, respectively. The derivative information in form of Jacobian-vector products is obtained by automatic differentiation. We investigate the performance impact of the two methods with respect to several algorithmic parameters such as the CFL number and the choice of the preconditioner.

The structure of this paper is as follows. In the following two sections, the formulation of the Euler- and Navier-Stokes equations as well as the solver QUADFLOW used throughout this work are briefly described. In Section 4 we give a brief introduction to automatic differentiation and discuss the actual implementation of the Jacobian-vector product in QUADFLOW. In Section 5, numerical experiments for inviscid and viscous flow problems are reported.

2 Governing Equations

In the present study, laminar viscous fluid flow is described by the Navier-Stokes equations for a compressible gas. Neglecting viscous effects, we obtain the Euler equations. The conservation laws for any control volume V with boundary ∂V and outward unit normal vector $\mathbf{n}$ on the surface element $dS \subset \partial V$ can be written in integral form as

$$\int_V \frac{\partial \mathbf{u}}{\partial t}\, dV + \oint_{\partial V} \left(\mathbf{F}^c(\mathbf{u}) - \mathbf{F}^d(\mathbf{u}) \right) \mathbf{n}\, dS = \mathbf{0}\,,$$

where a stationary grid is assumed. To complete the problem formulation, initial values $\mathbf{u}(\mathbf{x}, t_0) = \mathbf{u}_0(\mathbf{x})$, $\mathbf{x} \in V$ and boundary conditions $\mathbf{u}(\mathbf{x}, t)|_{\partial V} = B(\mathbf{x}, t)$, $\mathbf{x} \in \partial V$ are to be prescribed. Here, $\mathbf{u} = (\varrho, \varrho\mathbf{v}^T, \varrho e_{\text{tot}})^T$ denotes the vector of the unknown conserved quantities, and $\mathbf{F}^c$ and $\mathbf{F}^d$ represent the convective flux including pressure and the diffusive flux function, respectively. That is,

$$\mathbf{F}^c = \begin{pmatrix} \varrho\mathbf{v} \\ \varrho\mathbf{v} \circ \mathbf{v} + p\,\mathbf{I} \\ \varrho e_{\text{tot}}\mathbf{v} + p\mathbf{v} \end{pmatrix}, \quad \mathbf{F}^d = \begin{pmatrix} 0 \\ \mathbf{T}^v \\ \mathbf{v}\mathbf{T}^v - \mathbf{q} \end{pmatrix},$$

where ϱ denotes the density, p is the static pressure, $\mathbf{v}$ is the velocity vector of the fluid, and e_{tot} represents the total energy. The symbol $\circ$ denotes the dyadic product, and $\mathbf{I}$ is the identity. The viscous stress tensor $\mathbf{T}^v$ for an isentropic Newtonian fluid is defined as

$$\mathbf{T}^v = \mu \cdot \left(\operatorname{grad}\mathbf{v} + (\operatorname{grad}\mathbf{v})^T\right) - \frac{2}{3}\mu \cdot (\operatorname{div}\mathbf{v})\,\mathbf{I}\,.$$

Heat conduction is modeled by Fourier's law $\mathbf{q} = -\lambda \operatorname{grad} T$, where the thermal conductivity is assumed as $\lambda = c_p\mu/Pr$, with Prandtl number $Pr = 0.72$. The molecular viscosity μ as a function of the temperature T is determined by the Sutherland formula [50]. The static pressure is related to the specific internal energy according to the equation of state for a perfect gas $p = \varrho(\gamma - 1)\left(e_{tot} - 1/2\,|\mathbf{v}|^2\right)$, where γ is the ratio of specific heats, which is taken as γ=1.4 for air.

3 Finite volume scheme

The flow computations in this study are performed using QUADFLOW, which solves the Euler- and Navier-Stokes equations for compressible fluid flow in two and three space dimensions. We give preference to quadrilateral and hexahedral meshes since they are widely accepted to facilitate best boundary fitted meshes for viscous fluid flow. A key idea is to represent such meshes with as few parameters as possible while successive refinements can be efficiently computed based on the knowledge of these parameters. This concept is embedded in a multiblock framework. The mesh in each block results from evaluating a parametric mapping from the computational domain into the physical domain. Such mappings are based on B-Spline representations [32]. The mesh is locally adapted to the solution according to the concept of h-adaptation. The adaptation strategy gives rise to locally refined meshes of quadtree respectively octree type. A key role is played by reliable and efficient refinement strategies. In QUADFLOW, the adaptation criteria are based on recent multiresolution techniques [39, 22, 13]. Finally, a cell centered finite volume scheme that meets the requirements of the adaptive technique completes the concept. It is designed to cope with fairly general cell partitions and allows, in particular, to handle hanging nodes in a unified manner. The locally adaptive mesh is treated as a fully unstructured mesh, composed of simply connected elements with otherwise arbitrary topology.

3.1 Discretization of Inviscid Fluxes

The discretization of the inviscid fluxes is based on upwind methods. In the present study, the robust flux-vector splitting proposed by Hänel and Schwane [25] is employed for solving the Euler equations. For solving the Navier-Stokes equations, flux-vector splitting methods are too diffusive. In this case we employ the HLLC flux-difference splitting according to Batten et al. [5], which also resolves contact discontinuities.

The higher-order extension of the scheme is crucial to obtain accurate solutions of the governing equations. To obtain second-order accuracy in space, a linear reconstruction of the primitive flow variables $w \in \{\varrho, \mathbf{v}, p\}$ is defined as follows

$$w\left(\mathbf{x}\right)|_{V_i} := w_i + \phi_i \left(\mathbf{x} - \mathbf{x}_i\right)^T \cdot \nabla w_i \, , \, \mathbf{x} \in V_i$$

Here, w_i represents the solution at the centroid $\mathbf{x}_i$ of the control volume V_i, and ϕ_i denotes a limiter function, with $\phi_i \in [0, 1]$. To approximate the gradient ∇w_i of the quantity in question, either a least-squares technique or the Green-Gauss method may be employed. At local extrema and discontinuities the reconstruction polynomial may generate new extrema and therefore cause oscillations in the numerical solution. In order to circumvent this phenomenon, the slope limiter by Venkatakrishnan [54] is employed. To summarize the ideas from [54], let w_i^{min} and w_i^{max} denote the minimum respectively maximum values of w in the cell centers of all surrounding cells that support the reconstruction. For cell i, w_g denotes the unlimited reconstructed value of w at a Gauss point associated with one of the bounding faces. With $\tilde{w} \in \{w_i^{min}, w_i^{max}\}$, $\Delta_+ := \tilde{w} - w_i$, and $\Delta_- := w_g - w_i$ an auxiliary function ϕ is defined as follows

$$\phi\left(\frac{\Delta_+}{\Delta_-}\right) := \frac{\Delta_+^2 + 2\Delta_-\Delta_+ + \varepsilon}{\Delta_+^2 + 2\Delta_-^2 + \Delta_-\Delta_+ + \varepsilon} \, , \quad \text{with } \varepsilon \in [10^{-3}, 10^{-6}]$$

Defining

$$\phi_{i,g} := \begin{cases} \phi\left(\frac{\Delta_+}{\Delta_-}\right) & \text{with } \tilde{w} := w_i^{max}, \quad \text{if } \Delta_- > 0 \\ \phi\left(\frac{\Delta_+}{\Delta_-}\right) & \text{with } \tilde{w} := w_i^{min}, \quad \text{if } \Delta_- < 0 \\ 1, & \text{if } \Delta_- = 0 \end{cases}$$

for every quadrature point of the faces that bound the control volume, the Venkatakrishnan limiter is taken as the corresponding minimum value

$$\phi_i = \min_g(\phi_{i,g}) \, .$$

The function ϕ_i is only partially differentiable. A fully differentiable alternative to the Venkatakrishnan limiter has been introduced by Rosendale [48]. Rosendale proposes a generalized formulation of van Albada's limiter [52] that circumvents the use of $\min(\cdot)$ and $\max(\cdot)$ functions. Kemath et al. [27] and Casteleiro et al. [18] propose two limiters which are based on Rosendale's development.

However, the comparison between Venkatakrishnan's and van Albada's limiter shows that the latter results in a more diffusive scheme. Venkatakrishnan formulated both limiters in the framework by Spekreijse [49]. For a structured

grid the quantity w_g at a face between the cells i and $i+1$ is denoted by $w_{i+1/2}$, which in Spekreijse's framework reads as follows

$$w_{i+1/2} = w_i + \frac{w_{i+1} - w_{i-1}}{2\Delta x}\frac{\Delta x}{2}\Psi \ .$$

Here, Ψ is the limiter function ϕ_i formulated in Spekreijse's framework. It is assumed that w_i^{min} and w_i^{max} are given by w_{i-1} and w_{i+1}, respectively, and that w_i is bounded between w_{i-1} and w_{i+1}. Herewith, the functions Ψ_{vk} for the Venkatakrishnan limiter and Ψ_{va} for the van Albada limiter can be written as follows:

$$\begin{aligned} \Psi_{vk}(r) &= \frac{1}{2}(r+1)\min\left[\frac{4r(3r+1)}{11r^2+4r+1}, \frac{4(r+3)}{r^2+4r+11}\right] , \\ \Psi_{va}(r) &= \frac{r+r^2}{r^2+1} , \end{aligned} \tag{1}$$

where r denotes the ratio of slopes, i.e., $r = \dfrac{w_{i+1} - w_i}{w_i - w_{i-1}}$.

In Fig. 1, both limiters (1) are depicted as a function of the ratio r. The comparison shows clearly, that the Venkatakrishnan limiter allows the use of larger slope-ratios in the reconstruction process than the van Albada limiter and thus provides a higher spatial accuracy. Therefore we prefer the Venkatakrishnan limiter despite of its lack of strict differentiability.

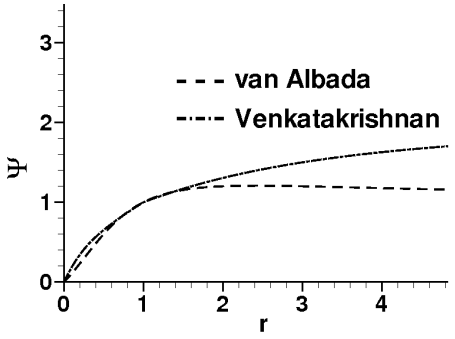

Figure 1: Limiter functions of van Albada and Venkatakrishnan

3.2 Discretization of Diffusive Fluxes

For the discretization of the diffusive fluxes, the gradients of the velocity vector, $\nabla \mathbf{v}_i$, and the temperature, ∇T, are required at the cell interfaces. The simplest procedure is to compute the gradients of the quantity in question, ∇w, within each cell and then average ∇w between the two cells that share a face on its left-hand side (∇w_L) and on its right-hand side (∇w_R) respectively. That is, the gradient ∇w at the cell interface is approximated by

$$\overline{\nabla w}\big|_{face} = \frac{1}{2}\left(\nabla w_L + \nabla w_R\right) . \tag{2}$$

The gradients ∇w_L and ∇w_R are provided by the (unlimited) reconstruction procedure. This kind of discretization supports undamped oscillatory modes that result from an odd-even point decoupling. A tighter coupling of the solution is obtained by approximating the gradient at the face in the direction $\mathbf{l}_{LR} = \mathbf{x}_R - \mathbf{x}_L$, which connects the centroids $\mathbf{x}_L$ and $\mathbf{x}_R$ of the left and right cell, respectively, by the divided difference

$$\left.\frac{\partial w}{\partial \mathbf{l}_{LR}}\right|_{face} = \frac{w_R - w_L}{|\mathbf{l}_{LR}|}\,. \tag{3}$$

Finally, the gradient is expressed by combining Eq. (2) and Eq. (3), i.e.,

$$\nabla w|_{face} = \overline{\nabla w}|_{face} - \left(\overline{\nabla w}|_{face} \cdot \frac{\mathbf{l}_{LR}}{|\mathbf{l}_{LR}|} - \frac{w_R - w_L}{|\mathbf{l}_{LR}|}\right)\frac{\mathbf{l}_{LR}}{|\mathbf{l}_{LR}|}\,.$$

3.3 Implicit Time Integration

After applying the spatial discretization, we obtain a system of ordinary differential equations

$$\int_V \frac{\partial \mathbf{u}}{\partial t}\,dV + \mathbf{R}\,(\mathbf{u}) = \mathbf{0}\,,$$

where $\mathbf{R}\,(\mathbf{u})$ denotes the residual vector defined by the sum of the discretized fluxes. To compute stationary flows, the first-order accurate implicit Euler method is chosen since time accuracy is not required to reach steady state in the computation. The resulting system of nonlinear equations is expressed as

$$\widehat{\mathbf{R}} := \left(\mathbf{u}^{n+1} - \mathbf{u}^n\right)\frac{|V|}{\Delta t} + \mathbf{R}(\mathbf{u}^{n+1}) = \mathbf{0}\,. \tag{4}$$

The solution $\mathbf{u}^{n+1}$ of the nonlinear system (4) is determined by a Newton iteration within each physical time step:

$$\widehat{\mathbf{J}}(\mathbf{u}^{(\ell)})\Delta\mathbf{u}^{(\ell)} = -\widehat{\mathbf{R}}(\mathbf{u}^{(\ell)}) \tag{5}$$

with

$$\lim_{\ell\to\infty} \mathbf{u}^{(\ell)} = \mathbf{u}^{n+1}.$$

Here, $\Delta\mathbf{u}^{(\ell)} := \mathbf{u}^{(\ell+1)} - \mathbf{u}^{(\ell)}$ denotes the change of the solution within each Newton step, indicated by the superscript (ℓ). The Jacobian of the system of equations, $\widehat{\mathbf{J}}(\mathbf{u}^{(\ell)})$, contains contributions of the temporal discretization and the spatial discretization, i.e.,

$$\widehat{\mathbf{J}}(\mathbf{u}^{(\ell)}) = \frac{\partial\widehat{\mathbf{R}}(\mathbf{u}^{(\ell)})}{\partial\mathbf{u}^{(\ell)}} = \mathbf{J}_t(\mathbf{u}^{(\ell)}) + \mathbf{J}(\mathbf{u}^{(\ell)})$$

with

$$\mathbf{J}_t(\mathbf{u}^{(\ell)}) = \frac{|V|}{\Delta t}\,\mathbf{I} \quad \text{and} \quad \mathbf{J}(\mathbf{u}^{(\ell)}) = \frac{\partial\mathbf{R}(\mathbf{u}^{(\ell)})}{\partial\mathbf{u}^{(\ell)}}\,.$$

The initial guess is $\mathbf{u}^{(0)} = \mathbf{u}^n$. For stationary flows, one Newton iteration to solve (5) is sufficient since convergence to steady state is enforced by the nonlinear (time) iteration. In this case, the Newton scheme for the implicit Euler time integration reduces to

$$\left(\frac{|V|}{\Delta t}\,\mathbf{I} + \frac{\partial \mathbf{R}(\mathbf{u}^n)}{\partial \mathbf{u}^n}\right) \cdot \Delta \mathbf{u}^n = -\mathbf{R}(\mathbf{u}^n)\ . \tag{6}$$

Note that, for time step sizes approaching infinity, this results in a pure Newton scheme. To enhance convergence to steady state, a local time step within each cell is chosen with a constant CFL number in the domain.

4 Computation of the Jacobian-vector product

Implicit time integration schemes, based on Newton's method, require the solution of the linear system of equations according to (6). The linear system is solved by a Krylov subspace method. Krylov subspace methods do not require the system matrix $\widehat{\mathbf{J}}(\mathbf{u})$ in an explicit form but only the product of $\widehat{\mathbf{J}}(\mathbf{u})$ with a Krylov vector $\mathbf{z}$. For a recent survey on Jacobian-free Newton-Krylov methods we refer to [31]. The Jacobian-vector product $\widehat{\mathbf{J}}(\mathbf{u}) \cdot \mathbf{z}$ can be split in two parts. The contribution of the temporal operator is simply

$$\mathbf{J}_t(\mathbf{u}) \cdot \mathbf{z} = |V| / \Delta t \cdot \mathbf{z}\ .$$

The complicated part is the computation of the Jacobian-vector product of the spatial operator:

$$\mathbf{J}(\mathbf{u}) \cdot \mathbf{z} = \frac{\partial \mathbf{R}(\mathbf{u})}{\partial \mathbf{u}} \cdot \mathbf{z}\ .$$

A popular approach to compute such Jacobian-vector products is approximation by divided differences, e.g.,

$$\mathbf{J}(\mathbf{u}) \cdot \mathbf{z} \approx \frac{\mathbf{R}(\mathbf{u} + \varepsilon \mathbf{z}) - \mathbf{R}(\mathbf{u})}{\varepsilon}\ . \tag{7}$$

This approach is used, e.g., in [17, 41, 44, 34, 45, 33]. While easy to implement, divided differencing has the conceptual disadvantage that the approximation quality depends on the value of the perturbation parameter ε, which is typically not known a priori. On the one hand, ε should be small in order to minimize the truncation error that results from truncating the Taylor series expansion. On the other hand, ε should not be *too* small because otherwise the rounding error due to cancellation effects becomes dominant when the numerator in (7) is evaluated in finite precision arithmetic.

For a first-order approximation a simple rule of thumb is to choose ε about the square root of the machine precision. Other approaches for choosing ε take into account the typical size of $\mathbf{u}$ or the precision in the computation of $\mathbf{R}$; see, for instance, [21, 19, 15, 28, 36, 24, 35]. Note that the problem of determining a suitable value for ε persists when higher-order approximations, e.g., central differences, are employed.

In contrast to numerical differentiation methods based on divided differencing a set of techniques referred to as automatic differentiation can be used to

compute derivative information without truncation error. Automatic differentiation can be applied whenever the function to be differentiated is given in form of a computer program. There are several software tools available, implementing automatic differentiation for various programming languages, including Fortran, C/C++ and MATLAB; see `www.autodiff.org` for an overview. The key idea is that any mathematical function given in the form of a computer program is composed of elementary arithmetic operations such as binary addition, multiplication, etc., and a limited set of intrinsic functions for which the derivatives are known. By applying the chain rule of differential calculus to the composition of elementary operations the derivative of the overall function can be computed more accurately than any approach using divided differences. Moreover, the computational complexity to compute a Jacobian-vector product by the so-called forward mode of automatic differentiation is in the same order as for a first-order divided difference approximation. That is, while computing the Jacobian-vector product by first-order forward divided differencing requires twice as many arithmetic operations as a single computation of the residual function, this factor is typically between 2 and 3 for automatic differentiation.

As an example, consider the following Fortran code fragment implementing some function $\mathbb{R}^n \to \mathbb{R}^n$.

```
      t=0.d0
      do i=1,n
         t = t + sin(x(i))
         y(i) = t * x(i)
      enddo
```

Using the automatic differentiation tool ADIFOR [6], this code fragment is automatically transformed to compute not only the original function but also the corresponding derivative information.

```
        g_t = 0.0d0
        t = 0.d0
C--------
        do i = 1, n
          g_t = cos(x(i)) * g_x(i) + g_t
          t = t + sin(x(i))
C--------
          g_y(i) = t * g_x(i) + x(i) * g_t
          y(i) = t * x(i)
C--------
        enddo
```

When the array variable `g_x` is initialized with some vector $\mathbf{z} \in \mathbb{R}^n$, the above code fragment computes the original function values, $\mathbf{y}$, as well as the Jacobian-vector product, $(\partial \mathbf{y}/\partial \mathbf{x}) \cdot \mathbf{z}$, and stores the corresponding values in the program variables `y` and `g_y`, respectively. We stress that automatic differentiation is not restricted to small computer codes, but scales well for large industrial software packages consisting of several hundreds of thousands of lines of source code [8]. For additional information on automatic differentiation the reader is referred to the books [46, 23] and the proceedings of recent workshops [16, 7].

4.1 Implementation in QUADFLOW

Approximate and exact Newton methods

In a previous work [12] the automatic differentiation tool ADIFOR has been employed in QUADFLOW to compute a lower-order approximation of the Jacobian $\mathbf{J} = \mathbf{J}_{\text{low}}$ of the higher-order operator $\mathbf{R}_{\text{high}}$. More precisely, the linearization of the fluxes is based on a first-order accurate method in space, which means that the linearization takes into account only direct neighboring cells. Hence, the linear system (6) is given by

$$\widehat{\mathbf{J}}_{\text{low}} \Delta \mathbf{u} = -\mathbf{R}_{\text{high}} \ . \tag{8}$$

Note that the approximation $\widehat{\mathbf{J}} = \widehat{\mathbf{J}}_{\text{low}}$ yields an approximate Newton method rather than an exact one. Consequently we cannot expect quadratic convergence behavior. For details on the implementation we refer to [12]. In essence, the ADIFOR tool was used to automatically transform given Fortran source code for the elementary flux functions of various upwind schemes into Fortran code for evaluating the corresponding local Jacobians. The overall Jacobian $\widehat{\mathbf{J}}_{\text{low}}$ is then constructed in a block-wise fashion, where the local Jacobians are appropriately inserted into a global data structure. In [12] it was shown that the use of automatic differentiation in the computation of the local Jacobians not only yields a more robust and reliable overall computational scheme, compared to an approach based on divided differences, but can also improve the rate of convergence significantly and is faster in terms of CPU time.

Inspired by these results, we employ automatic differentiation to implement the exact linearization of the higher-order method, i.e.,

$$\widehat{\mathbf{J}}_{\text{high}} \Delta \mathbf{u} = -\mathbf{R}_{\text{high}} \ . \tag{9}$$

In contrast to equation (8), equation (9) represents an exact Newton scheme with the advantage of quadratic asymptotic convergence. Since the memory requirement for storing $\mathbf{J}_{\text{high}}$ is about three times as large as for $\mathbf{J}_{\text{low}}$, we avoid the explicit construction of $\mathbf{J}_{\text{high}}$, but rather use automatic differentiation to evaluate the matrix-vector products during the iterative solution of the system (9).

Implementation details

In the actual implementation, the differentiation procedure could not be conducted in a fully automatic manner, since QUADFLOW consists of modules written in different programming languages. The high-level control flow of the program is implemented in the C programming language, while the mathematical core routines are written in Fortran. Currently no automatic differentiation tool is capable of augmenting mixed-language programs with derivatives. Therefore we follow a semi-automatic approach: First, the low-level Fortran routines are transformed by ADIFOR. In the current implementation, the automatic generation of derivative code using ADIFOR is fully integrated into the build process of the QUADFLOW executable. That is, the code transformation is automatically performed on the fly during the compilation process. This procedure facilitates that any modification of the mathematical core, for example the solution of the Riemann problem, is always consistently reflected within the Jacobian-vector product without the need of manual intervention. In a second

step, the higher-level C routines are manually modified in such a way, that the interfaces to the Fortran subroutines are consistent with the according differentiated version. In particular, the interfaces are extended by the additional gradient information. No further modification of the higher-level routines is required, as long as the interfaces to the low-level subroutines remain unchanged.

One may raise the question how we apply automatic differentiation to piecewise differentiable intrinsic functions, such as $\min(\cdot)$ or $\max(\cdot)$, which are used in the implementation of the Venkatakrishnan limiter [54]. As an example, consider the following code fragment taken from the actual implementation in QUADFLOW to determine the minimum value `wmin` of the `i`-th component of the solution vector `w` of the cell `icell` in its adjacent neighborhood `iNeighbor`

```
wmin(i,icell) = min(wmin(i,icell),w(i,iNeighbor))
```

The derivative may be undefined if `wmin(i,icell)` equals `w(i,iNeighbor)`. ADIFOR provides a flexible exception handling mechanism that not only reports such situations, but also allows the user to specify the exceptional behavior. In practice it is often desired to continue the derivative computation with some reasonable value, even if the derivative is not defined in a strict sense. In the case of the binary $\min(\cdot)$ or $\max(\cdot)$ intrinsics, according to the generalized gradient principle, such a reasonable value is the average of the partial derivatives of the two arguments. This is ADIFOR's default strategy to handle non-differentiability in these intrinsics. To our practical experience, this strategy is also appropriate for the computation of derivatives in QUADFLOW. For the example above, the following code is generated by ADIFOR[1]:

```
      d3_v = min(wmin(i,icell), w(i,iNeighbor))
      if (wmin(i,icell) .lt. w(i,iNeighbor)) then
         d1_p = 1.0d0
         d2_p = 0.0d0
      else if (wmin(i,icell) .gt. w(i,iNeighbor)) then
         d1_p = 0.0d0
         d2_p = 1.0d0
      else
         d1_p = 0.5d0
         d2_p = 0.5d0
      endif
      g_wmin(i,icell) =   d2_p * g_w(i,iNeighbor)
     &                  + d1_p * g_wmin(i,icell)
      wmin(i,icell) = d3_v
```

The linear systems (8) respectively (9) are solved using the BiCGSTAB algorithm [53], combined with left preconditioning, which is based on a point-block-ILU(p) method [43], denoted by PBILU(p), where p is the level of fill-in. The implementation of the Newton-Krylov method is based on the PETSc [1,2] library developed at Argonne National Laboratory. In the present study, we still construct the matrix $\widehat{\mathbf{J}}_{\text{low}}$ in its explicit form to determine its incomplete lower-upper factorization, PBILU(p). Thus, the current approach is not completely matrix-free. However, the preconditioning matrix has to be constructed only

[1] ADIFOR is executed in the `performance` mode

once for every linear system, and not for each Krylov subspace iteration. This approach of employing a low-order approximation of the Jacobian for purpose of preconditioning in a higher-order method has been successfully used by many researchers; see, for instance, [55,51,9,44,33,20,35,40]. In a recent article, Wong and Zingg [56] compare the memory requirement, CPU cost, and effectiveness of various ILU preconditioners based on $\widehat{\mathbf{J}}_{\text{low}}$ as well as $\widehat{\mathbf{J}}_{\text{high}}$, where the latter is found to be computationally too expensive. Other preconditioning techniques, that do not require the Jacobian in its explicit form, are investigated in [34,45].

Hybrid Newton method

Although the exact Newton method has the advantage of local quadratic convergence, it is unnecessary or even counterproductive in the startup phase of the computation as it can diverge in practical applications, e.g., in the transonic regime. That is, if the starting point is too far away from the solution, Newton's method may fail. To circumvent this problem, we apply an approximate Newton method, based on the first-order accurate Jacobian $\mathbf{J}_{\text{low}}$, during the early iterations. When the relative residual of the density first drops below a prescribed threshold ν, we switch to the exact Newton method using $\mathbf{J}_{\text{high}}$. For timestep k let R denote the relative residual of the density, i.e.,

$$R := \frac{\|\mathbf{R}_k\|_1}{\|\mathbf{R}_0\|_1} \,. \tag{10}$$

With k_ν representing the first time step satisfying $R \leq \nu$ the actual implementation is as follows.

$$\left.\begin{array}{rcl} \widehat{\mathbf{J}}_{\text{low}}\Delta\mathbf{u} & = & -\mathbf{R}_{\text{high}}\,, \text{ if } k \leq k_\nu \\ \widehat{\mathbf{J}}_{\text{high}}\Delta\mathbf{u} & = & -\mathbf{R}_{\text{high}}\,, \text{ if } k > k_\nu \end{array}\right\} \tag{11}$$

A similar strategy is employed in [9,35]. Other authors suggest to switch after an a-priori prescribed number of time steps [42,40].

5 Numerical experiments

In this section we present results of numerical experiments comparing the performance of the Newton-Krylov method employing approximate Jacobian-vector products, which are based on a first-order accurate discretization in space, and exact Jacobian-vector products, based on a second-order accurate discretization in space. For simplicity, the method employing the approximate Jacobian-vector products is called "first-order method", whereas the exact Newton method is called "second-order method". For the first-order method, we usually compute the Jacobian-vector product by relying on an explicit representation of the Jacobian, $\mathbf{J}_{\text{low}}$, because this matrix is assembled anyway for the purpose of preconditioning. A matrix-free implementation of the first-order method is available for performance comparison. For the second-order method we always use the matrix-free implementation.

All computations are carried out with QUADFLOW, which is executed on an Intel Xeon quad-core processor running at 3 GHz clock speed. In Section 5.1 we present performance results for the simulation of the inviscid flow around two

standard profiles (test cases 1 and 2). In Section 5.2 an inviscid flow around a swept wing is considered (test case 3). As a final test case (test case 4) the two-dimensional laminar viscous flow over a flat plate is considered in Section 5.3.

The linear systems resulting from Newton's method are solved until the relative residual is less than 10^{-2} for the Euler equations, and less than 10^{-3} for the Navier-Stokes equations. [2]

5.1 Test problems 1 and 2: 2D flow around NACA0012 and BAC 3-11/RES/30/21 airfoils

We consider the inviscid transonic stationary flow around the NACA0012 airfoil [29], a standard test case for validating CFD codes, and a configuration assembled from the airfoil system BAC 3-11/RES/30/21 [38], which is used as reference configuration within the collaborative research center SFB 401 [3]. The corresponding flight conditions are given in Table 1. As an example, the locally adapted grid and the Mach distribution for test case 1B are presented in Fig. 2.

Table 1: Test cases 1A, 1B, 1C, and 2: Mach number M_∞ and angle of attack α for NACA0012 airfoil and BAC 3-11/RES/30/21 airfoil.

NACA0012	M_∞	α
test case 1A	0.80	1.25°
test case 1B	0.95	0°
test case 1C	1.20	0°

BAC 3-11/RES/30/21	M_∞	α
test case 2	0.77	0°

The numerical experiments in this section are carried out using a variant of the implicit Euler method, namely the b2-scheme [5]. On every level of grid refinement, except the finest level, the time integration is performed until the relative residual of the density, R, is less than 10^{-2}. On the finest level we require $R \leq 10^{-10}$. The local time step is determined by an exponential rule for the CFL number. That is, the CFL number in time step k is given by $CFL_k = CFL_{FAC} \cdot CFL_{k-1}$, where CFL_{FAC} is a constant factor, and $CFL_0 = 1$. The CFL number is limited by a fixed upper bound, CFL_{MAX}. We allow 8 levels of refinement, i.e., each cell can be subdivided isotropically at most 8 times. In the following we present results for a restart on the finest grid. We carried out 10 adaptations in test cases 1A, 1C, and 2. In test case 1B we perform 14 adaptations.

As mentioned above, the linearized systems of equations are preconditioned by a PBILU(p) method. For reasons discussed later, we set the number of ILU

[2] We observed that using a lower accuracy might speed up the overall method in some cases, but also often diverges towards a non-physical state. On the other hand, solving the linear systems with higher accuracy significantly increases the number of Krylov iterations without actually decreasing the number of time steps.

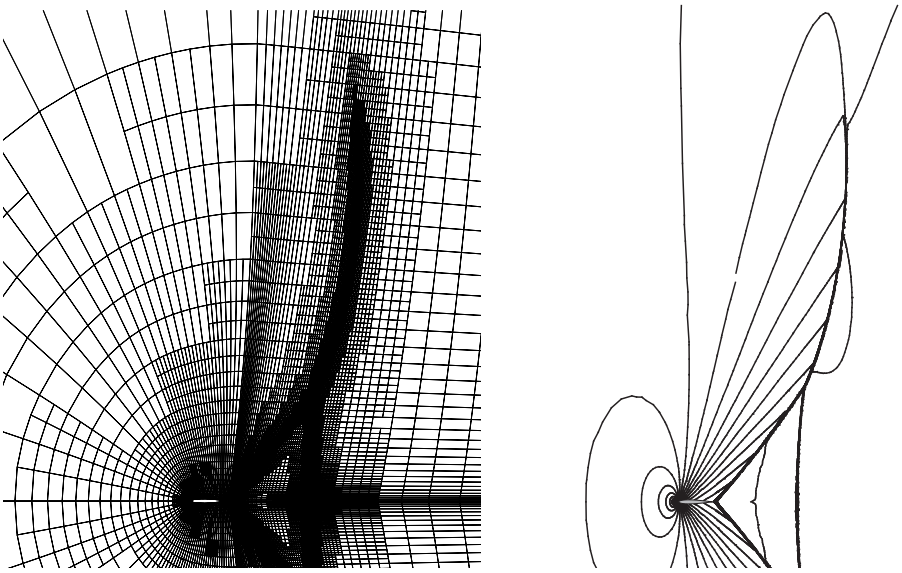

Figure 2: Test case 1B: Detail of the finest computational grid consisting of 34552 cells (left) and the corresponding Mach distribution (right). Mach numbers are given between $M_{min} = 0.0$ and $M_{max} = 1.45$, $\Delta M = 0.05$.

levels to $p = 0$ for the first-order methods and use $p = 2$ for the second-order method.

In Table 2 we present results for two different CFL evolution strategies for test case 1A. The first, rather cautious, evolution strategy is characterized by the parameters $CFL_{FAC} = 1.2$ and $CFL_{MAX} = 10^2$. The second, more aggressive, strategy increases the CFL numbers faster, with a factor $CFL_{FAC} = 1.5$ and applies an upper bound $CFL_{MAX} = 10^6$. The number of time steps, denoted by $\#ts$, and the corresponding execution times are given for the first-order method and the second-order method. In addition, the hybrid method (11) with the switch parameter $\nu = 10^{-5}$ is investigated. From Table 2 we observe that, if the cautious CFL evolution strategy is used, the number of time steps required for the second-order method is less than for the first-order method. If the more aggressive evolution strategy is used, the second-order method fails, whereas the hybrid method reduces the number of time steps compared to the first-order method. However, the second-order and hybrid methods require more CPU time than the first-order method in test case 1A.

In test cases 1B and 1C, the aggressive CFL strategy is used, whereas a more cautious strategy with $CFL_{FAC} = 1.2$ and $CFL_{MAX} = 10^5$ is used in test case 2 in order to avoid divergence to a non-physical state during the startup phase. In Table 3 the number of time steps and the corresponding CPU times are given for the matrix-based first-order method, the matrix-free first-order method, and the hybrid method with switch parameter $\nu = 10^{-5}$. The number of time steps is almost identical for both first-order implementations, while the hybrid method needs significantly fewer iterations. Comparing the execution times for the first-order methods it turns out that the matrix-based implementation is always

Table 2: Test case 1A: Number of time steps #ts and CPU time for the first-order, second-order, and hybrid methods using two different CFL evolution strategies.

		first-order		second-order		hybrid ($\nu = 10^{-5}$)	
CFL_{FAC}	CFL_{MAX}	# ts	CPU [s]	# ts	CPU [s]	# ts	CPU [s]
1.2	10^2	2265	750.4	2190	3798.6	2201	3686.7
1.5	10^6	82	66.2	–	–	56	111.7

faster than the matrix-free variant. This is caused by the fact that the matrix is explicitly built for preconditioning in both cases. Clearly, the computation of a single matrix-vector product using an explicit representation of the Jacobian matrix is generally faster than a matrix-free evaluation. Nevertheless, when using the hybrid method, the overall execution times are reduced by 43% in test case 1B and 16% in test case 2, compared to the first-order matrix-based method. In test case 1C, no performance improvement can be observed.

Table 3: Test cases 1B, 1C, and 2: Number of time steps #ts and CPU time for the first-order matrix-based, first-order matrix-free, and hybrid methods. The parameters for the CFL strategy are $CFL_{FAC} = 1.5, CFL_{MAX} = 10^6$ in test cases 1B and 1C, and $CFL_{FAC} = 1.2, CFL_{MAX} = 10^5$ in test case 2.

	first-order matrix-based		first-order matrix-free		hybrid ($\nu = 10^{-5}$)	
Test case	# ts	CPU [s]	# ts	CPU [s]	# ts	CPU [s]
1B	539	503.1	534	576.8	61	287.9
1C	74	158.3	74	163.9	51	167.5
2	101	595.8	101	672.6	70	500.4

The corresponding residual histories with respect to the time step and the CPU time are given in the left and right columns of Fig. 3, respectively. More precisely, the residual histories for the first-order method and the hybrid method with switch parameters $\nu = 10^{-2}$, $\nu = 10^{-5}$, and $\nu = 10^{-6}$, are presented. It can be observed from the left column of Fig. 3 that the rate of convergence immediately increases when, in the hybrid method, the switch from first-order to second-order takes place. This can lead to a significant reduction of the overall execution time of the simulation, if the switch parameter ν is chosen appropriately; see the right column of Fig. 3. However, if the switch to the second-order method is performed too early, i.e., the parameter ν is chosen too large, the execution time increases.

There are several reasons for the computational overhead of the second-order method compared to the first-order method. Obviously, more arithmetic operations are performed in the second-order method. In addition, as mentioned above, the Jacobian $\widehat{\mathbf{J}}_{\text{low}}$, which is assembled in both cases, is used to compute the matrix-vector product explicitly in the first-order method. Moreover, solving the linear systems is more difficult, if the exact Jacobian matrix is used.

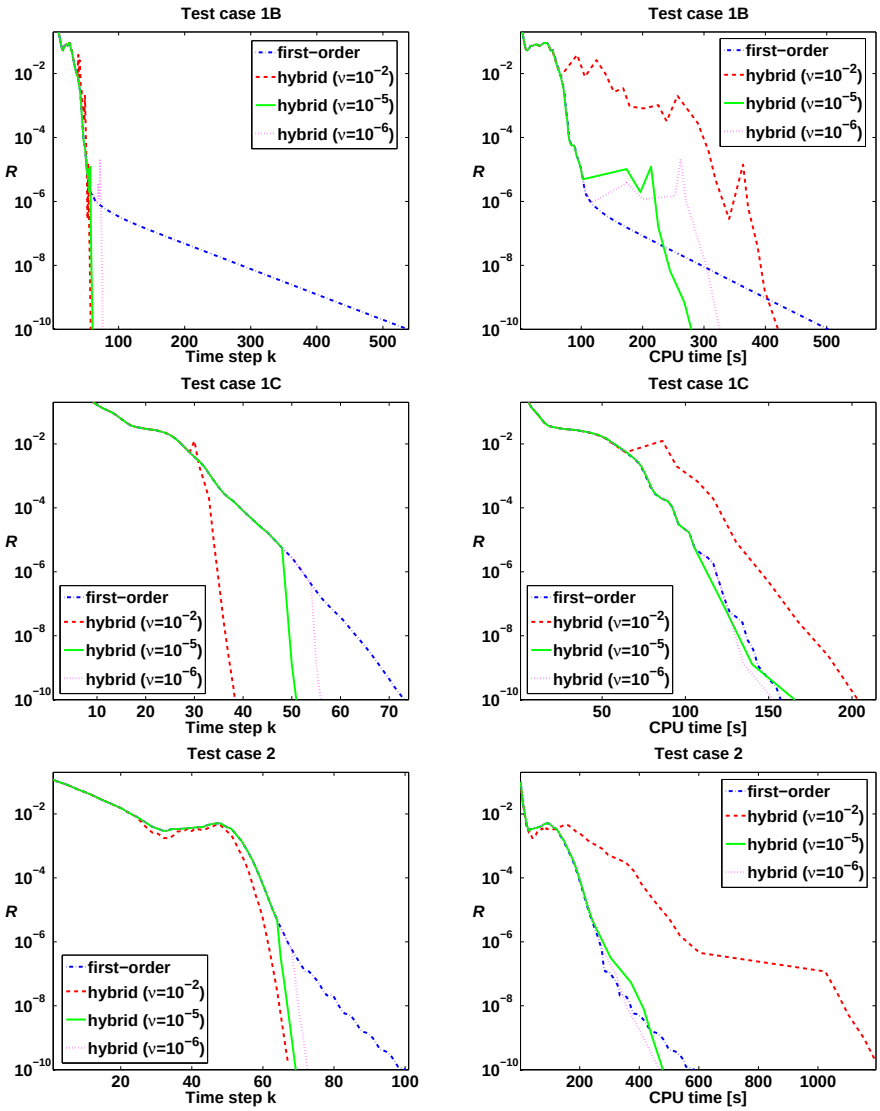

Figure 3: Test cases 1B (upper row), 1C (middle row), and 2 (lower row): Residual history in terms of iterations (left subplots) and CPU time (right subplots).

To illustrate this, the iteration history of the BiCGSTAB solver is presented in Fig. 4. For every time step, the number of BiCGSTAB iterations using the PBILU(p) preconditioner with $p \in \{0, 1, 2\}$ is presented for the first-order method (left subplot) and the second-order method (right subplot). The iteration counts for $p = 1$ and $p = 2$ are higher for the second-order method than for the first-order method. For $p = 0$, the linear system of equations could not be solved using the second-order method.

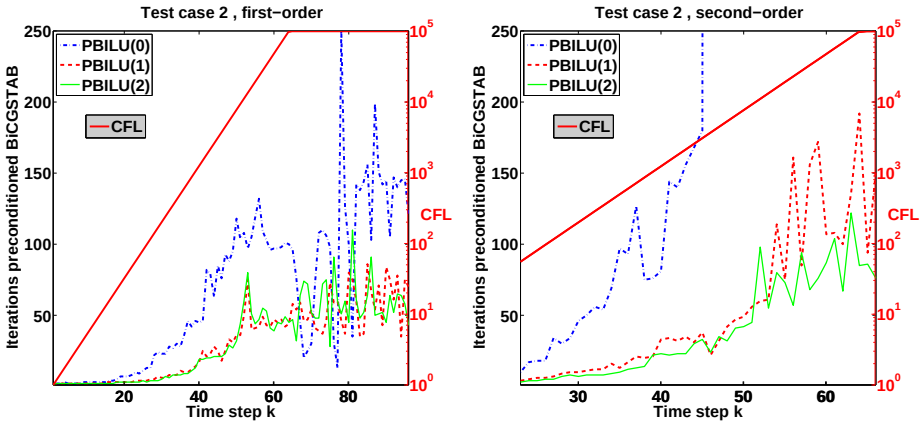

Figure 4: Test case 2: BiCGSTAB iteration history for the second step of the b2-scheme using different PBILU(p) preconditioners, $p = 0, 1, 2$. The corresponding CFL numbers are illustrated by the bold red line with a logarithmic scale. The first-order method (left subplot) is compared to the final phase of the hybrid method when the second-order method is employed (right subplot). The switch parameter is $\nu = 10^{-2}$.

5.2 Test problem 3: 3D flow around swept wing

In this section the computation of the subsonic inviscid flow around a swept wing of constant chord length with the BAC 3-11/RES/30/21 profile in cruise configuration and a sweep angle $\varphi = 34.0°$ is considered; see Fig. 5. This configuration has been designed and experimentally investigated within the collaborative research center SFB 401 [47, 30]. The flow parameters are $M_\infty = 0.22$ and $\alpha = 4.64°$. The implicit Euler scheme is used for the time integration. The non-adaptive, multiblock structured grid consists of 425984 cells. The computation is performed in parallel using 4 MPI tasks.

We compare the number of time steps and the execution time of the first-order method and the hybrid method with switch parameter $\nu = 10^{-2}$. Since the startup phase is identical for both methods, we focus on the final phase of the computation, i.e., $k > k_\nu$ in (11). The solution is considered to be converged when the relative residual R is less than 10^{-4}. In Table 4 we present performance results for different values of constant CFL numbers, $CFL \in \{10^2, 10^3, 10^4\}$. For each CFL number we vary the parameter $p \in \{1, 2, 3\}$.

From Table 4 it can be observed that, for large CFL numbers, increasing the value of p can lead to a reduction of the overall execution time. However, the cor-

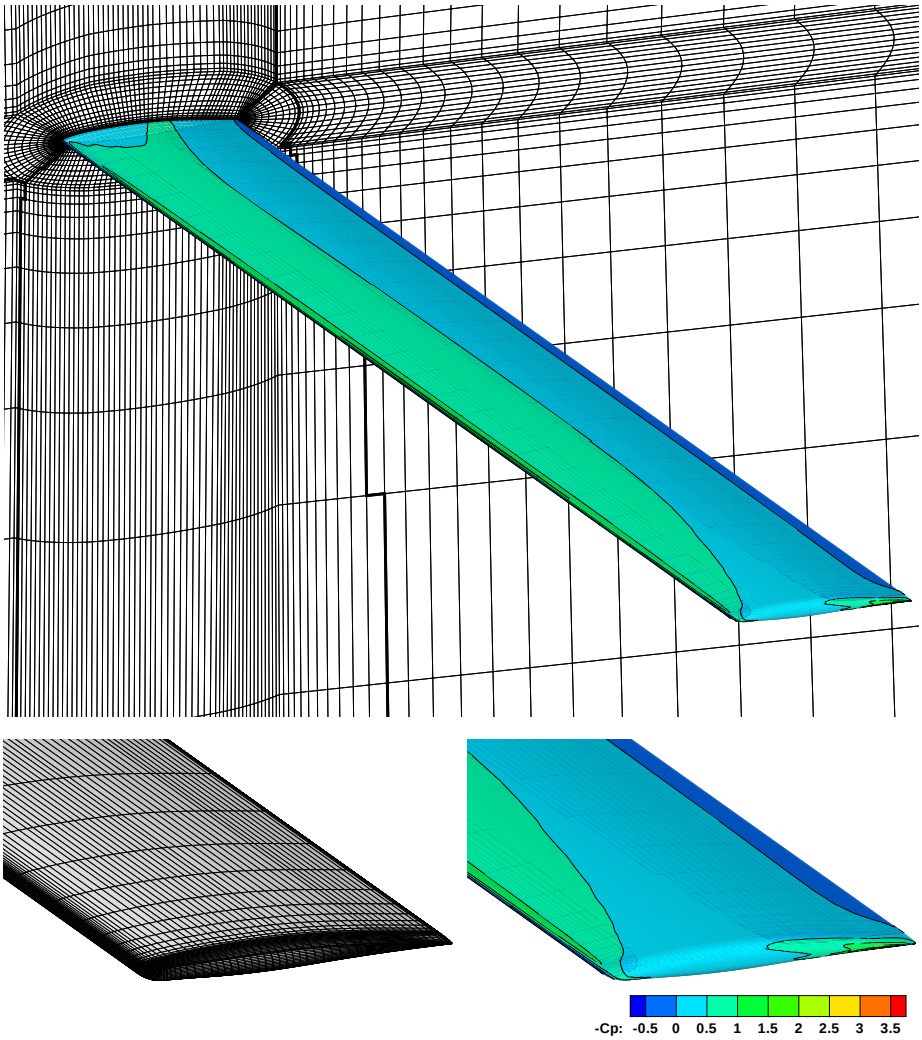

Figure 5: Text case 3: Swept wing. Upper plot: contour plot of pressure distribution. Lower plot: details of the wing tip.

responding increased memory requirements have to be considered. The number of time steps is significantly decreased for both, the first- and the second-order method, if the CFL number is increased. The corresponding execution times increase for the first-order method and decrease for the second-order method. For $CFL = 10^4$, the second-order method is significantly faster than the first-order method. On the other hand, with $CFL = 10^2$, the first-order method is faster. Note that for small values of CFL, the higher cost for a single Krylov iteration slows down the second-order method, whereas for large CFL numbers the time integration benefits from the higher accuracy in the Newton step. In the latter case, i.e., $CFL = 10^4$, the execution time of the second-order method is significantly less than for all first-order computations.

Table 4: Test case 3: Number of time steps #ts and CPU time for different values for CFL and different PBILU(p) preconditioners, neglecting the startup phase.

		first-order		second-order	
CFL	p	# ts	CPU [s]	# ts	CPU [s]
10^2	1	142	1488.2	164	3672.7
	2	142	1583.4	92	3747.5
	3	142	2136.6	92	4159.3
10^3	1	77	1575.1	12	1858.5
	2	77	1614.7	12	1493.2
	3	77	1904.5	12	1468.4
10^4	1	71	1902.9	5	674.1
	2	71	1859.5	4	558.5
	3	71	2097.5	4	620.5

5.3 Test problem 4: 2D laminar flow over a flat plate

In the following we consider the laminar viscous flow over a flat plate. The freestream parameters are $M_\infty = 0.2$, $Re = 10^4$, and $T_\infty = T_{wall} = 273K$. The non-adaptive multiblock structured grid consists of two blocks with 64 cells each in the direction normal to the wall. In flow direction, the domain is discretized with 200 and 100 cells, respectively, in the block containing the plate and the block upstream of the plate.

The spatial discretization of the convective fluxes is based on the HLLC flux difference splitting, and for the time integration, the implicit Euler scheme is chosen. We compare the first-order method and the hybrid method with $\nu = 10^{-1}$. As in test case 3, we focus on the final part of the computation, neglecting the startup phase. The solution is considered to be converged when $R \leq 10^{-6}$. The number of time steps and the corresponding CPU times for different constant values for CFL and p are listed in Table 5.

Table 5: Test case 4: Number of time steps #ts and CPU time for different values for CFL and different PBILU(p) preconditioners, neglecting the startup phase.

		first-order		second-order	
CFL	p	# ts	CPU [s]	# ts	CPU [s]
10^2	2	924	299.8	700	657.4
	4	924	330.1	700	653.2
	6	924	370.7	699	685.5
10^4	2	409	352.8	13	108.2
	4	409	326.6	13	87.8
	6	409	315.2	13	82.6
10^6	2	412	380.3	7	55.0
	4	412	338.2	7	40.9
	6	412	323.0	7	38.4

When increasing the value for CFL from 10^2 to 10^6, the number of time steps for the second-order method is reduced by a factor of 100. This leads to a significant decrease of the corresponding CPU time. Contrarily, this behavior cannot be observed for the first-order method. As remarked in the previous section, larger values for p can further decrease the CPU time.

For the first-order method the "best" configuration in terms of CPU time is characterized by the parameter setting $p = 2$ and $CFL = 10^2$. The second-order method is faster by a factor of 7.8, when the parameter setting $p = 6$ and $CFL = 10^6$ is chosen. If a smaller value for p is desired, the second-order method is faster by a factor of 5.5, when using the parameter setting $p = 2$ and $CFL = 10^6$.

6 Conclusion

In this paper we describe the implementation of an implicit matrix-free Newton-Krylov method for solving the Euler- and Navier-Stokes equations. The mathematical core of the algorithm consists of the solution of a system of linear equations involving the Jacobian of the spatial operator. Due to memory limitations, in many practical implementations only an approximation of the Jacobian based on a first-order accurate discretization in space is computed. However, this approach yields an approximate Newton method rather than an exact one. The explicit assembling of the Jacobian matrix can be avoided by means of automatic differentiation. Utilizing this technology we implemented the computation of the exact Jacobian-vector product for a second-order accurate discretization in space.

Several numerical experiments for stationary inviscid and viscous compressible fluid flow were conducted. It was demonstrated that the use of the exact Jacobian-vector product can significantly reduce the number of nonlinear iterations, i.e., time steps. However, when the exact Jacobian is used, a single time step consumes more CPU time as if the approximate Jacobian were considered. Thus, a hybrid strategy is suggested, which employs the approximate Newton method during the startup phase and switches later to the exact Newton method.

The current implementation of the second-order method still requires the explicit Jacobian based on a first-order accurate discretization for preconditioning. Future plans include the replacement of this matrix-based preconditioner by some kind of matrix-free preconditioner.

Acknowledgments

This work has been performed with funding by the Deutsche Forschungsgemeinschaft (DFG) within the collaborative research center SFB 401 "Flow Modulation and Fluid-Structure Interaction at Airplane Wings" at RWTH Aachen University, Aachen, Germany. The authors acknowledge the fruitful collaboration with the other members of the QUADFLOW research group. We also thank Lars Reimer for providing the computational grid for the swept wing.

References

[1] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory (2004).

[2] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A. M. Bruaset, H. P. Langtangen (eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997.

[3] J. Ballmann (ed.), Flow Modulation and Fluid-Structure-Interaction at Airplane Wings, vol. 84 of Notes on Numerical Fluid Mechanics and Multidisciplinary Design, Springer, Berlin, 2003.

[4] T. J. Barth, S. W. Linton, An unstructured mesh Newton solver for compressible fluid flow and its parallel implementation, AIAA Paper 1995–0221.

[5] P. Batten, M. A. Leschziner, U. C. Goldberg, Average-state Jacobians and implicit methods for compressible viscous and turbulent flows, Journal of Computational Physics 137 (1) (1997) 38–78.

[6] C. Bischof, A. Carle, P. Khademi, A. Mauer, Adifor 2.0: Automatic differentiation of Fortran 77 programs, IEEE Computational Science & Engineering 3 (3) (1996) 18–32.

[7] C. H. Bischof, H. M. Bücker, P. D. Hovland, U. Naumann, J. Utke (eds.), Advances in Automatic Differentiation, vol. 64 of Lecture Notes in Computational Science and Engineering, Springer, Berlin, 2008.

[8] C. H. Bischof, H. M. Bücker, A. Rasch, E. Slusanschi, B. Lang, Automatic differentiation of the general-purpose computational fluid dynamics package FLUENT, Journal of Fluids Engineering 129 (5) (2007) 652–658.

[9] M. Blanco, D. W. Zingg, Fast Newton-Krylov method for unstructured grids, AIAA Journal 36 (4) (1998) 607–612.

[10] K. H. Brakhage, S. Müller, Algebraic-hyperbolic grid generation with precise control of intersection of angles, International Journal for Numerical Methods in Fluids 33 (1) (2000) 89–123.

[11] F. D. Bramkamp, Unstructured h-adaptive finite-volume schemes for compressible viscous fluid flow, Ph.D. thesis, RWTH Aachen University (2003).

[12] F. D. Bramkamp, H. M. Bücker, A. Rasch, Using exact Jacobians in an implicit Newton-Krylov method, Computers & Fluids 35 (10) (2006) 1063–1073.

[13] F. D. Bramkamp, B. Gottschlich-Müller, M. Hesse, P. Lamby, S. Müller, J. Ballmann, K. H. Brakhage, W. Dahmen, H-Adaptive Multiscale Schemes for the Compressible Navier–Stokes Equations — Polyhedral Discretization, Data Compression and Mesh Generation, in: Ballmann [3], vol. 84 of Notes on Numerical Fluid Mechanics and Multidisciplinary Design, pp. 125–204.

[14] F. D. Bramkamp, P. Lamby, S. Müller, An adaptive multiscale finite volume solver for unsteady and steady state flow computations, Journal of Computational Physics 197 (2) (2004) 460–490.

[15] P. N. Brown, Y. Saad, Hybrid Krylov methods for nonlinear systems of equations, SIAM Journal on Scientific and Statistical Computing 11 (3) (1990) 450–481.

[16] H. M. Bücker, G. F. Corliss, P. D. Hovland, U. Naumann, B. Norris (eds.), Automatic Differentiation: Applications, Theory, and Implementations, vol. 50 of Lecture Notes in Computational Science and Engineering, Springer, Berlin, 2005.

[17] T. F. Chan, K. R. Jackson, Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms, SIAM Journal on Scientific Computing 5 (3) (1984) 533–542.

[18] L. Cueto-Felgueroso, I. Colominas, X. Nogueira, F. Navarrina, M. Casteleiro, Finite volume solvers and moving least-squares approximations for the compressible Navier-Stokes equations on unstructured grids, Computer Methods in Applied Mechanics and Engineering 196 (45–48) (2007) 4712–4736.

[19] J. E. Dennis, R. B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, NJ, USA, 1983.

[20] P. Geuzaine, Newton-Krylov strategy for compressible turbulent flows on unstructured meshes, AIAA Journal 39 (3) (2001) 528–531.

[21] P. E. Gill, W. Murray, M. H. Wright, Practical Optimization, Academic Press, New York, 1981.

[22] B. Gottschlich-Müller, Multiscale schemes for conservation laws, Ph.D. thesis, RWTH Aachen University (1998).

[23] A. Griewank, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM, Philadelphia, PA, 2000.

[24] I. M. G. Grotowsky, J. Ballmann, Efficient time integration of Navier-Stokes equations, Computers & Fluids 28 (2) (1999) 243–263.

[25] D. Hänel, R. Schwane, An implicit flux–vector splitting scheme for the computation of viscous hypersonic flow, AIAA Paper 1989–0274.

[26] P. D. Hovland, L. C. McInnes, Parallel simulation of compressible flow using automatic differentiation and PETSc, Parallel Computing 27 (4) (2001) 503–519.

[27] P. Jawahar, H. Kemath, A high-resolution procedure for Euler and Navier-Stokes computations on unstructured grids, Journal of Computational Physics 164 (1) (2000) 165–203.

[28] Z. Johan, T. J. R. Hughes, F. Shakib, A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids, Computer Methods in Applied Mechanics and Engineering 87 (2–3) (1991) 281–304.

[29] D. J. Jones, Reference test cases and contributors, in: AGARD–AR–211: Test Cases for Inviscid Flow Field Methods, Advisory Group for Aerospace Research & Development, Neuilly-sur-Seine, France, 1986.

[30] M. Kämpchen, A. Dafnis, H.-G. Reimerdes, Aero-structural response of a flexible swept wind tunnel wing model in subsonic flow, in: Proceedings of the CEAS AIAA/NVVL International Forum on Aeroelasticity and Structural Dynamics, IFASD 2003, Amsterdam, The Netherlands, 2003.

[31] D. A. Knoll, D. E. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, Journal of Computational Physics 193 (2) (2004) 357–397.

[32] P. Lamby, Parametric multi-block grid generation and application to adaptive flow simulations, Ph.D. thesis, RWTH Aachen University (2007).

[33] I. Lepot, P. Geuzaine, F. Meers, J.-A. Essers, J.-M. Vaassen, Analysis of several multigrid implicit algorithms for the solution of the Euler equations on unstructured meshes, in: E. Dick, K. Riemslagh, J. Vierendeels (eds.), Multigrid Methods VI: Proceedings of the Sixth European Multigrid Conference, Gent, Belgium, September 27–30, 1999, vol. 14 of Lecture Notes in Computational Science and Engineering, Springer, Berlin, 2000.

[34] H. Luo, J. D. Baum, R. Löhner, A fast, matrix-free implicit method for compressible flows on unstructured grids, Journal of Computational Physics 146 (2) (1998) 664–690.

[35] L. Manzano, J. V. Lassaline, P. Wong, D. W. Zingg, A Newton-Krylov algorithm for the Euler equations using unstructured grids, AIAA Paper 2003–0274.

[36] P. R. McHugh, D. A. Knoll, Comparison of standard and matrix-free implementations of several Newton-Krylov solvers, AIAA Journal 32 (12) (1994) 394–400.

[37] A. Meister, Comparison of different Krylov subspace methods embedded in an implicit finite volume scheme for the computation of viscous and inviscid flow fields on unstructured grids, Journal of Computational Physics 140 (2) (1998) 311–345.

[38] I. R. M. Moir, Measurements on a two-dimensional aerofoil with high-lift-devices, in: AGARD–AR–303: A Selection of Experimental Test Cases for the Validation of CFD Codes, volume 1 and 2, Advisory Group for Aerospace Research & Development, Neuilly-sur-Seine, France, 1994.

[39] S. Müller, Adaptive Multiscale Schemes for Conservation Laws, vol. 27 of Lecture Notes on Computational Science and Engineering, Springer, Berlin, 2002.

[40] A. Nejat, C. Ollivier-Gooch, Effect of discretization order on preconditioning and convergence of a high-order unstructured Newton-GMRES solver for the Euler equations, Journal of Computational Physics 227 (4) (2008) 2366–2386.

[41] E. J. Nielsen, W. K. Anderson, R. W. Walters, D. E. Keyes, Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code, AIAA Paper 1995–1733.

[42] F. Olawsky, F. Infed, M. Auweter-Kurtz, Preconditioned Newton-Method for computing supersonic and hypersonic nonequilibrium flows, Journal of Spacecraft and Rockets 41 (6) (2004) 907–914.

[43] B. Pollul, Preconditioners for linearized discrete compressible Euler equations, in: P. Wesseling, E. Oñate, J. Périaux (eds.), Proceedings of the European Conference on Computational Fluid Dynamics ECCOMAS, Egmond aan Zee, The Netherlands, 2006.

[44] A. Pueyo, D. W. Zingg, Efficient Newton-Krylov solver for aerodynamic computations, AIAA Journal 36 (11) (1998) 1991–1997.

[45] N. Qin, D. K. Ludlow, S. T. Shaw, A matrix-free preconditioned Newton/GMRES method for unsteady Navier-Stokes solutions, International Journal for Numerical Methods in Fluids 33 (2) (2000) 223–248.

[46] L. B. Rall, Automatic Differentiation: Techniques and Applications, vol. 120 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 1981.

[47] L. Reimer, C. Braun, J. Ballmann, Analysis of the static and dynamic aero-structural response of an elastic swept wing model by direct aeroelastic simulation, in: Proceedings of the 25th International Congress of the Aeronautical Sciences, ICAS 2006, Hamburg, Germany, 2006.

[48] J. V. Rosendale, Floating shock fitting via Lagrangian adaptive meshes, Tech. Rep. 94–89, ICASE (1989).

[49] S. P. Spekreijse, Multigrid solution of monotone second-order discretizations of hyperbolic conservation laws, Math. Comput. 49 (1987) 135–155.

[50] W. Sutherland, The viscosity of gases and molecular force, Philosophical Magazine 36 (5) (1893) 507–531.

[51] M. D. Tidriri, Preconditioning techniques for the Newton-Krylov solution of compressible flows, Journal of Computational Physics 132 (1) (1997) 51–61.

[52] G. D. van Albada, B. van Leer, W. W. Roberts, A comparative study of computational methods in cosmic gas dynamics, Astronomy and Astrophysics 108 (1) (1982) 76–84.

[53] H. A. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 13 (2) (1992) 631–644.

[54] V. Venkatakrishnan, Convergence to steady state solutions of the Euler equations on unstructured grids with limiters, Journal of Computational Physics 118 (1) (1995) 120–130.

[55] V. Venkatakrishnan, D. J. Mavriplis, Implicit solvers for unstructured meshes, Journal of Computational Physics 105 (1) (1993) 83–91.

[56] P. Wong, D. W. Zingg, Three-dimensional aerodynamic computations on unstructured grids using a Newton-Krylov approach, Computers & Fluids 37 (2) (2008) 107–120.