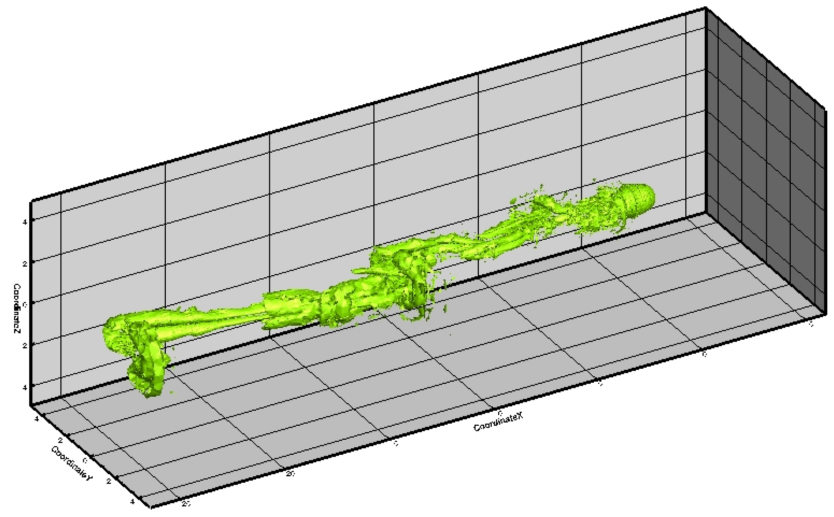
Numerical Methods for the Unsteady Compressible Navier-Stokes Equations



Philipp Birken

Numerical Methods for the Unsteady Compressible Navier-Stokes Equations

Dr. Philipp Birken

Habilitationsschrift
am Fachbereich Mathematik und Naturwissenschaften
der Universität Kassel

Gutachter:	Prof. Dr. Andreas Meister
Zweitgutachterin:	Prof. Dr. Margot Gerritsen
Drittgutachterin:	Prof. Dr. Hester Bijl

Probevorlesung: 31. 10. 2012

There are all kinds of interesting questions that come from a knowledge of science, which only adds to the excitement and mystery and awe of a flower. It only adds. I don't understand how it subtracts.

Richard Feynman

Acknowledgements

A habilitation thesis like this is not possible without help from many people. First of all, I'd like to mention the German Research Foundation (DFG), which has funded my research since 2006 as part of the collaborative research area SFB/TR TRR 30, project C2. This interdisciplinary project has provided me the necessary freedom, motivation and optimal funding to pursue the research documented in this book. It also allowed me to hire students to help me in my research and I'd like to thank Benjamin Badel, Maria Bauer, Veronika Diba, Marouen ben Said and Malin Trost, who did a lot of coding and testing and created a lot of figures. Furthermore, I thank the colleagues from the TRR 30 for a very successful research cooperation and in particular Kurt Steinhoff for starting the project.

Then, there are my collaborators over the last years, in alphabetical order Hester Bijl, who thankfully also acted as reviewer, Jurjen Duintjer Tebbens, Gregor Gassner, Mark Haas, Volker Hannemann, Stefan Hartmann, Antony Jameson, Detlef Kuhl, Andreas Meister, Claus-Dieter Munz, Karsten J. Quint, Mirek Tuma and Alexander van Zuijlen. Furthermore, thanks to Mark H. Carpenter and Gustaf Söderlind for a lot of interesting discussions.

I'd also like to thank the University of Kassel and the Institute of Mathematics, as well as the colleagues there for a friendly atmosphere. In particular, there are the colleagues from the Institute of Mathematics in Kassel: Thomas Hennecke, Gunnar Matthies, Bettina Messerschmidt, Sigrun Ortleb who also did a huge amount of proofreading, Martin Steigemann, Philipp Zardo and special thanks to my long time office mate Stefan Kopecz.

Furthermore, there are the colleagues at Stanford at the Institute of Mathematical and Computational Engineering and the Aero-Astro department, where I always encounter a friendly and inspiring atmosphere. In particular, I'd like to thank the late Gene Golub, Michael Saunders and Antony Jameson for many interesting discussions and getting things started. Furthermore, I'd like to thank Margot Gerritsen for being tremendously helpful and taking on the second review.

My old friend Tarvo Thamm for proofreading.

Not the last and certainly not the least, there is my former PhD advisor Andreas Meister and reviewer of this habilitation thesis, who has organized my funding and supported and enforced my scientific career whereever possible.

Finally, there is my Margit who had to endure this whole process as much as me and who makes things worthwhile.

Contents

Chapter 1

Introduction

Historically, the computation of steady flows has been at the forefront of the development of computational fluid dynamics (CFD). This began with the design of rockets and the computation of the bow shock at supersonic speeds and continued with the aerodynamic design of airplanes at transonic cruising speed [99]. Only in the last decade, increasing focus has been put on unsteady flows, which are more difficult to compute. This has several reasons. First of all, computing power has increased dramatically and for 5,000 Euro it is now possible to obtain a machine that is able to compute about a minute of realtime simulation of a nontrivial unsteady three dimensional flow in a day. As a consequence, ever more nonmilitary companies are able to employ numerical simulations as a standard tool for product development, opening up a large number of additional applications. Examples are the simulation of tunnel fires [19], flow around wind turbines [225], fluid-structure-interaction like flutter [59], aeroacoustics [40], turbomachinery, flows inside nuclear reactors [164], wildfires [163], hurricanes and unsteady weather phenomenas [162], gas quenching [126] and many others. More computing capacities will open up further possibilities in the next decade, which suggests that the improvement of numerical methods for unsteady flows should start in earnest now. Finally, the existing methods for the computation of steady states, while certainly not at the end of their development, have matured, making the consideration of unsteady flows interesting for a larger group of scientists. This research monograph is written with both mathematicians and engineers in mind, to give them an overview of the state of the art in the field and to discuss a number of new developments, where we will focus on the computation of laminar viscous flows, as modelled by the Navier-Stokes equations.

These are the most important model in fluid dynamics, from which a number of other widely used models can be derived, for example the incompressible Navier-Stokes equations, the Euler equations or the shallow water equations. An important feature of fluids that is present in the Navier-Stokes equations is turbulence, which roughly speaking appears if the Reynolds number of the problem at hand is large enough. Since Reynolds numbers for air flows as modelled by the compressible equations are large, most airflows of practical importance are turbulent. However, turbulence adds a completely new level of complexity to all aspects of solver design. Therefore, we will focus on laminar flows instead, which are more accessible by a mathematical perspective.

Another important feature of the Navier-Stokes equations is the boundary layer, which makes it necessary to use very fine grids. Since explicit time integration methods have an inherent stability constraint, they need to choose their time step on these grids based on stability only and not via error control. This makes the use of implicit time integration methods desirable, since these are not bound by stability constraints as explicit schemes. However, using implicit scheme requires solving linear or nonlinear equation systems. Consequently, a large part of this book will be about implicit schemes and schemes for solving linear and nonlinear equation systems.

1.1 The method of lines

Here, we follow the method of lines paradigm:

1. The space discretization transforms the partial differential equation (PDE) into a system of ordinary differential equations (ODE), introducing a discretization error.

2. The implicit time integration transforms the ODE into a system of algebraic equations, introducing a time integration error, that needs to be related to the discretization error.

3. If this algebraic system is nonlinear, it is approximately solved by an iterative method (until termination), which introduces an iteration error that needs to be smaller than the time discretization error.

4. If this algebraic system is linear or if Newton's method was chosen in the previous step, another iterative method solves the linear system (until termination), which introduces another iteration error, which needs to related to the time discretization error, respectively the error in the nonlinear solver.

The main advantage of the method of lines is its immense flexibility, allowing the reuse of existing methods that can be tested and analyzed for the simpler subproblems, as well as the simple exchange of any part of the overall solution procedure. Therefore, the use of the method of lines is ubiquitious in scientific computing, meaning that most of the techniques discussed here can be applied to a large class of problems, as long as they are modelled by time dependent partial differential equations. On the other hand, the main drawback of the approach is in the limitations of how spatial and temporal adaptation can be connected. Therefore, space-time discretizations are an interesting alternative.

Thus, we arrive at the common questions of how we can guarantee robustness and accuracy, obtain efficient methods and all of that at reasonable implementation cost?

Regarding space discretization, the most prominent methods for flow problems are finite volume methods. These respect a basic property of the Navier-Stokes equations, namely conservation of mass, momentum and energy and, despite significant theory gaps for multidimensional problems, it is known how to obtain robust schemes of order up to two. The question of obtaining higher order schemes is significantly more difficult and has been an extremely active area of research for the past ten years. As the most prominent type of methods there, discontinuous Galerkin (DG) schemes have been established. These use higher order polynomials and a Galerkin condition to determine the solution. However, as for all schemes of higher order, the question of efficiency for the Navier-Stokes equations is still the topic of much ongoing research, in particular for implicit time integration.

In the case of an explicit time integration, the best choice are explicit Runge-Kutta methods. In the implicit case, BDF methods are used widely in industry, essentially because these just need to solve one nonlinear system per time step. However, third and higher order BDF schemes have limited stability properties. Furthermore, time adaptivity is a problematic issue. This is important for an implicit scheme, since the whole point is to choose the time step based on error estimators and not on stability constraints. Therefore, singly diagonally implicit Runge-Kutta (SDIRK) schemes are the important alternative, which was shown to be competitive with BDF methods at engineering tolerances [17]. These consist of a sequence of backward Euler steps with different right hand sides and time step sizes, thus allowing a modular implementation. Furthermore, time adaptivity can be done at almost no additional cost using embedded schemes. Another interesting alternative, in particular for more accurate computations, are Rosenbrock schemes that in essence are linearized SDIRK methods, thus only needing linear system solves.

The steps that mainly determine the efficiency of an implicit time integration scheme are the last two in the method of lines: Solving equation systems. In the industry but also in academia, code development has been driven by the desire to compute steady flows. This requires the solution of one large algebraic system and the fastest codes to do so use a multigrid method. For this reason, the majority of codes used in industry employ this strategy, for example the DLR TAU-code [66]. The multigrid method for steady states can be carried over to unsteady flows using the dual time stepping approach [98]. Since this allows to compute unsteady flows at essentially no additional implementation cost, dual time stepping is the method of choice in the said codes.

The main alternative is Newton's method, which requires solving sequences of linear systems. Therefore, common data structures needed are vectors and matrices. Since an explicit code is typically based on cell or point based data structures and not on vectors, the implementation cost of this type of methods is considered to be high. Together with the fact that the canonical initial guess for the steady state (freestream data) is typically outside the region of convergence of Newton's method, this has led to a bad reputation of the method in the CFD community.

Now, if we consider a steady state equation, discretized in space, we obtain a nonlinear algebraic equation of the form

$$\underline{\mathbf{f}}(\underline{\mathbf{u}}) = \mathbf{0}$$

with $\underline{\mathbf{u}} \in \mathbb{R}^m$ and $\underline{\mathbf{f}} : \mathbb{R}^m \rightarrow \mathbb{R}^m$. However, the nonlinear equation arising from an implicit time discretization in the method of lines is of the form

$$(\underline{\mathbf{u}} - \underline{\mathbf{s}})/(\alpha\Delta t) - \underline{\mathbf{f}}(\underline{\mathbf{u}}) = \mathbf{0},$$

where $\underline{\mathbf{s}}$ is a given vector, α a method dependent parameter and Δt the time step size. Due to the additional term, we actually expect the second system to be easier to solve.

When considering the existing multigrid codes employing dual time stepping, it turns out that if at all, the convergence speed is increased only slightly. This can be explained quite easily: A multigrid method depends on the specific PDE to be solved, as well as on the discretization. If the PDE is changed, in this case by adding a time derivative term, we cannot expect the method to perform well. On the other hand this means that better multigrid methods can be designed and we will illuminate a route on how to do that.

The second important point regarding the change from steady to unsteady states is that for Newton's method in an implicit time integration, the canonical initial guess is the solution from the last time level. This is in a sense close to the solution at the new time level which means that the performance of Newton's method changes dramatically for the better when going from steady to unsteady flows.

Furthermore, when solving the linear equation systems using a Krylov subspace method like GMRES, the Jacobian is needed in matrix vector products only. Since the matrix is a Jacobian, it is possible to replace these by a finite difference approximation. Thus, a method is obtained that does not need a Jacobian and needs no additional implementation effort when changing the spatial discretization, just in the spirit of the flexibility of the method of lines. Unfortunately, this is not completely true in that Krylov subspace methods need a preconditioner to be truly efficient. It is here that a lot of research has been put in and more research needs to be done to obtain efficient, robust and easy to implement schemes. Summarizing, it is necessary to reevaluate and redesign the existing methods for unsteady flows.

As for preconditioning, the linear systems that arise have few properties that can be exploited, in particular they are nonnormal and not diagonally dominant for reasonable Δt. However, the systems have a block structure arising from grouping unknowns from the same cell together. For finite volume schemes, where the blocks are of size four for two dimensional flows and size five for three dimensional flows, a block incomplete LU (ILU) decomposition leads to a fast method. This situation is different for DG schemes. There, the number of unknowns in one cell depends on the dimension, the order of the scheme and the particular method chosen. As a rule of thumb, the number of unknowns in two dimensions is in the dozens and in the hundreds for three dimensional flows. This makes a significant difference for efficiency with the result being that currently, the class of problems where an implicit scheme is more efficient than an explicit one is significantly smaller for DG schemes than for FV schemes. For a specific DG scheme, namely a modal method with a hierarchical polynomial basis, we describe a preconditioner that is able to adress this specific problem.

Repeating the issues named earlier about robustness, efficiency and accuracy, an important topic arises: Since an implicit method requires one or two subiterations, there are a lot of parameters to choose in the method. Here, the goal is to have only one user defined parameter in the end, namely the error tolerance, and all other parameters should be determined from there. Furthermore, the termination criteria in the subsolver should be chosen such that there is not interference with the discretization error, but also no oversolving, meaning that these schemes should not perform more iterations than necessary. Finally, iterative methods have the inherent danger of divergence. Therefore, it is necessary to have feedback loops on what happens in these cases. Otherwise, the code will not be robust and therefore, not be used in practice. In these regards, mathematics is a tremendous help, since it allows to obtain schemes that are provable convergent, to derive termination criteria that are sharp and to detect divergence.

1.2 Hardware

A fundamental trend in computing since the introduction of the microprocessor has been Moore's law, meaning that the speed of CPUs increases exponentially. There is a second important trend, namely that instead of one computation on a single processor, multiple computations are carried out in parallel. This is true for huge clusters commonly used in high performance computing, for common PCs that have multiple CPUs with multiple cores and for GPUs that are able to handle a thousand threads in parallel. This means that numerical methods must be devised to perform well in parallel. Typically, this trend was driven by hardware manufacturers not having numerical computations in mind, then compilers and operating systems were built by computer scientists not having numerical computations in mind. This leads to a situation where the numerical analysts at the bottom of the food chain have a hard time designing optimal methods, in particular when hardware architecture is quickly changing as it is now.

However, there is an interesting development going on with graphics processing units (GPUs), which were originally designed for efficient single precision number crunching. When the market leader Nvidia realized that scientific computation is an attractive market, it developed GPUs able of performing double precision computations together with making coding on GPUs easier using the documentation of CUDA. This makes a programming paradigm, where codes are parallelized on multicore CPUs using the MPI standard and on top of that, GPUs are employed.

Another important trend is that the available memory per core will decrease in the future, meaning that computations will be not only compute bound, but also memory bound, at least for three dimensional calculations. Consequently, methods that use little memory are to be preferred.

1.3 Notation

Troughout this book, we will use bold capital letters ($\mathbf{A}$) to indicate matrices and bold small letters ($\mathbf{x}$) to indicate vectors. Small letters represent scalars, and thus the components of vectors are small letters with indices. Thus, $\mathbf{u}_1$ is the first vector of a family of vectors, but u_1 would be the first component of the vector $\mathbf{u}$. Specifically, the three space directions are x_1, x_2 and x_3 as the components of the vector $\mathbf{x}$. A vector with an underbar $\underline{\mathbf{u}}$ denotes a vector representing the discrete solution on the complete grid.

1.4 Outline

The book essentially follows the order of steps in the method of lines. We first describe in chapter 2 the basic mathematical models of fluid dynamics and discuss some of their properties. Then we will discuss space discretization techniques in chapter 3, in particular finite volume and discontinuous Galerkin methods. In chapter 4, different time integration techniques are presented, in particular explicit and implicit schemes, but also schemes that do not fall directly in the method of lines context. If implicit schemes are used, this results in linear or nonlinear equation systems and the solution of these is discussed in chapter 5. In particular, fixed point methods, multigrid methods, Newton methods and Krylov subspace methods are described, as well as Jabobian-Free Newton-Krylov methods. Preconditioning techniques for Krylov subspace methods are described in chapter 6. In chapter 7, we summarize and combine the techniques of the previous chapters and describe how the final flow solvers look like, as well give an assessment of their performance. Finally, unsteady thermal Fluid-Structure interaction is considered in chapter 8 and the application of the techniques discussed before is described.

Chapter 2

The Governing Equations

We will now describe the equations that will be used throughout this book. The mathematical models employed in fluid mechanics have their basis in continuum mechanics, meaning that it is not molecules that are described, but a large number of those that act as if they were a continuum. Thus velocities, pressure, density and similar quantities are of a statistical nature and say that on average, at a certain time, the molecules in a tiny volume will behave in a certain way. The mathematical model derived through this approach are the Navier-Stokes equations. The main component of these is the momentum equation, which was found in the beginning of the 19th century independently of each other by Navier [149], Stokes [187], Poisson [158] and de St. Venant [46]. During the course of the century, the equations were given more theoretical and experimental backing, so that by now, the momentum equation together with the continuity equation and the energy equation are established as the model describing fluid flow.

This derivation also shows a limitation of the mathematical model, namely for rarefied gases as in outer layers of the atmosphere, the number of molecules in a small volume is no longer large enough to allow statistics.

From the Navier-Stokes equations, a number of important simplifications have been derived, in particular the incompressible Navier-Stokes equations, the Euler equations, the shallow water equations or the potential flow equations. We will discuss in particular the Euler equations, which form an important basis for the development of discretization schemes for the Navier-Stokes equations, as well as an already very useful mathematical model in itself.

2.1 The Navier-Stokes Equations

The Navier-Stokes equations describe the behavior of a Newtonian fluid. In particular, they describe turbulence, boundary layers, as well as shock waves and other wave phenomenas.

They consist of the conservation laws of mass, momentum and energy. Thus they are derived from integral quantities, but for the purpose of analysis, they are often written in a differential form. There, they form a system of second order partial differential equations of mixed hyperbolic-parabolic type. If only the steady state is considered, the equations are elliptic-parabolic. A more detailed description can be found for example in the textbooks of Chorin and Marsden [37] and Hirsch [93].

In the following sections, we will start with dimensional quantities, denoted by the superscript $\hat{}$ and derive a nondimensional form later. We now consider an open domain $\mathcal{U} \subset \mathbb{R}^d$ and the elements of $\mathcal{U}$ are written as $\mathbf{x} = (x_1, ..., x_d)^T$.

2.1.1 Basic form of conservation laws

The conservation of a quantity is typically described by rewriting the amount $\phi_\Omega(t)$ given in a control volume $\Omega \subset \mathcal{U}$ using a local concentration $\psi(\mathbf{x}, t)$:

$$\phi_\Omega(t) = \int_\Omega \psi(\mathbf{x}, t) d\Omega.$$

Conservation means that the amount ϕ_Ω can only be changed by transport over the boundary of Ω or internal processes. An important tool to describe this change is Reynolds' transport theorem:

Theorem 1 (Reynolds' transport theorem) *Let $\Omega(t)$ be a possibly time dependent control volume, ψ a differentiable function and $\mathbf{v}(\mathbf{x}, t)$ be the velocity of the flow. Then*

$$\frac{d}{dt}\int_\Omega \psi d\Omega = \int_\Omega \partial_t \psi d\Omega + \int_{\partial\Omega} \psi \mathbf{v} \cdot \mathbf{n} ds. \tag{2.1}$$

The proof is straightforward using multidimensional analysis, see e.g. [222, page 10]. We will now consider control volumes that are fixed in time. Thus we have

$$\int_\Omega \partial_t \psi d\Omega = \partial_t \int_\Omega \psi d\Omega.$$

2.1.2 Conservation of mass

The *conservation equation of mass* (also called continuity equation) is given in terms of the density $\hat{\rho}$ for an arbitrary control volume Ω by

$$\partial_{\hat{t}} \int_\Omega \hat{\rho} d\Omega + \int_{\partial\Omega} \hat{\mathbf{m}} \cdot \mathbf{n} ds = 0. \tag{2.2}$$

Here $\hat{\mathbf{m}} = \hat{\rho}\hat{\mathbf{v}}$ denotes the momentum vector divided by the unit volume. Since this is valid for any Ω, application of the Gaussian theorem for the boundary integral leads to the differential form

$$\partial_{\hat{t}}\hat{\rho} + \nabla \cdot \hat{\mathbf{m}} = 0. \tag{2.3}$$

Here, the divergence operator is meant with respect to the spatial variables only, throughout the book.

Note that by contrast to the two following conservation laws, which are based on deep principles of theoretical physics like the Noether theorem, conservation of mass is not a proper law of physics, since mass can be destroyed and created. A particular example is radioactive decay, where mass is transformed into energy, meaning that the underlying physical law here is conservation of energy via Einsteins discovery of the equivalence of mass and energy. However, for the purpose of nonradioactive fluids at nonrelativistic speeds, (2.2) is a perfectly reasonable mathematical model.

2.1.3 Conservation of momentum

The equation for the *conservation of momentum* is based on Newton's second law, which states that the change of momentum in time is equal to the acting force. For the time being, we assume that there are no external forces acting on the fluid and look at surface forces only. As relevant terms we then have the pressure gradient, which results in a force, and the forces resulting from shear stresses due to viscous effects. Additionally we get the flow of momentum from Reynolds' transport theorem. With the pressure $\hat{p}$ and the velocity vector $\hat{\mathbf{v}}$, we obtain for an arbitrary control volume Ω

$$\partial_{\hat{t}} \int_\Omega \hat{m}_i d\Omega + \int_{\partial\Omega} (\hat{m}_i \hat{v}_j + \hat{p}\delta_{ij}) \cdot \mathbf{n} ds = \int_{\partial\Omega} \hat{\mathbf{S}} \cdot \mathbf{n} ds, \qquad i = 1, ..., d. \tag{2.4}$$

Again, application of the Gaussian theorem for the boundary integral leads to the differential form

$$\partial_{\hat{t}} \hat{m}_i + \sum_{j=1}^{d} \partial_{\hat{x}_j} (\hat{m}_i \hat{v}_j + \hat{p}\delta_{ij}) = \sum_{j=1}^{d} \partial_{\hat{x}_j} \hat{S}_{ij}, \qquad i = 1, ..., d, \tag{2.5}$$

where δ_{ij} is the Kronecker symbol and the viscous shear stress tensor $\mathbf{S}$ is given by

$$\hat{S}_{ij} = \hat{\mu} \left((\partial_{\hat{x}_j} \hat{v}_i + \partial_{\hat{x}_i} \hat{v}_j) - \frac{2}{3} \delta_{ij} \nabla \cdot \hat{\mathbf{v}} \right), \qquad i, j = 1, ..., d, \tag{2.6}$$

where $\hat{\mu}$ is the dynamic viscosity. In particular, this means that the shear stresses with $i \neq j$ are proportional to the velocity gradient. If this is not the case for a fluid, it is called non-Newtonian. Examples of this are fluids where the viscosity itself depends on the temperature or the velocity, namely blood, glycerin, oil or a large number of melted composite materials. Note that in (2.6), the experimentally well validated Stokes hypothesis is used, that allows to use only one parameter $\hat{\mu}$ for the description of viscosity.

2.1.4 Conservation of energy

Regarding *conservation of energy*, its mathematical formulation is derived from the first law of thermodynamics for a fluid. The first law states that the change in time in total energy in a control volume Ω is given by the flow of heat plus work done by the fluid. The heat flow is given by the convective flux from the Reynolds' transport theorem (2.1) plus the viscous flow due to Fourier's law of heat conduction. Furthermore, the work done by the fluid is due to the forces acting on it. Again, we neglect external forces for now and thus we have the pressure forces and the viscous stresses. With $\hat{E}$ being the total energy per unit mass we thus obtain for an arbitrary control volume:

$$\partial_{\hat{t}} \int_\Omega \hat{\rho}\hat{E} d\Omega + \int_{\partial\Omega} (\hat{H}\hat{\mathbf{m}}) \cdot \mathbf{n} ds = \int_{\partial\Omega} \sum_{j=1}^{2} \left(\sum_{i=1}^{2} \hat{S}_{ij}\hat{v}_i - \hat{W}_j \right) \cdot \mathbf{n} ds. \tag{2.7}$$

As before, application of the Gaussian theorem for the boundary integral leads to the differential form

$$\partial_{\hat{t}}\hat{\rho}\hat{E} + \nabla_{\hat{\mathbf{x}}} \cdot (\hat{H}\hat{\mathbf{m}}) = \sum_{j=1}^{d} \partial_{\hat{x}_j} \left(\sum_{i=1}^{d} \hat{S}_{ij}\hat{v}_i - \hat{W}_j \right). \tag{2.8}$$

$\hat{H} = \hat{E} + \frac{\hat{p}}{\hat{\rho}}$ denotes the enthalpy and $\hat{W}_j$ describes the flow of heat which, using the thermal conductivity coefficient $\hat{\kappa}$, can be written in terms of the gradient of the temperature $\hat{T}$ as

$$\hat{W}_j = -\hat{\kappa}\partial\hat{T}.$$

The total energy per unit mass $\hat{E}$ is given by the sum of inner and kinetic energy as

$$\hat{E} = \hat{e} + \frac{1}{2}|\hat{\mathbf{v}}^2|.$$

2.1.5 Equation of state

The five differential equations depend on the variables $\hat{\rho}$, $\hat{\mathbf{m}}$ and $\hat{\rho}\hat{E}$, but also on a number of others. These need to be determined to close the system. First of all, the thermodynamic quantities density, pressure and temperatur are related through the ideal gas law

$$\hat{T} = \frac{\hat{p}}{\hat{\rho}\hat{R}}. \tag{2.9}$$

Furthermore, we need an equation for the pressure $\hat{p}$, which is called the equation of state, since it depends on the particular fluid considered. Typically, it is given as a function $\hat{p}(\hat{\rho}, \hat{e})$. For an ideal gas and fluids similar to one, the adiabatic exponent γ can be used to obtain the simple form

$$\hat{p} = (\gamma - 1)\hat{\rho}\hat{e}, \tag{2.10}$$

which can be derived using theoretical physics. However, for some fluids, in particular some oils, the equation of state is not given as a function at all, but in the form of discrete measurings only.

Finally, the adiabatic exponent γ and the specific gas constant $\hat{R}$ are related through the specific heat coefficients for constant pressure $\hat{c}_p$, respectively constant volume $\hat{c}_v$, through

$$\gamma = \frac{\hat{c}_p}{\hat{c}_v}$$

and

$$\hat{R} = \hat{c}_p - \hat{c}_v.$$

For an ideal gas, γ is the quotient between the number of degrees of freedom plus two and the number of degrees of freedom. Thus, for a diatomic gas like nitrogen, $\gamma = 7/5$ and therefore, a very good approximation of the value of γ for air is 1.4. The specific gas constant is the quotient between the universal gas constant and the molar mass of the specific gas. For dry air, this results in $\hat{R} \approx 287$J/Kg/K. Correspondingly, we obtain $\hat{c}_p \approx 1010$J/Kg/K and $\hat{c}_v \approx 723$J/Kg/K.

2.2 Nondimensionalization

An important topic in the analysis of partial differential equations is the nondimensionalization of the physical quantities. This is done to achieve two things. First, we want all quantities to be $\mathcal{O}(1)$ due to stability reasons and then we want scalability from experiments to real world problems to numerical simulations. For the Navier-Stokes equations, we will obtain several reference numbers like the Reynolds number and the Prandtl number. These depend on the reference quantities and allow this scaling by specifying how reference values had to be changed to obtain solutions for the same Reynolds and Prandtl number.

A nondimensional form of the equations is obtained by replacing all dimensional quantities with the product of a nondimensional variable with a dimensional reference number:

$$\hat{\phi} = \phi \cdot \hat{\phi}_{ref}. \tag{2.11}$$

Given reference values for the variables length, velocity, pressure and density ($\hat{x}_{ref}$, $\hat{v}_{ref}$, $\hat{p}_{ref}$ and $\hat{\rho}_{ref}$), we can define the reference values for a string of other variables from these:

$$\hat{t}_{ref} = \frac{\hat{x}_{ref}}{\hat{v}_{ref}}, \qquad \hat{E}_{ref} = \hat{H}_{ref} = \hat{c}^2_{ref} = \frac{\hat{p}_{ref}}{\hat{\rho}_{ref}}, \qquad \hat{R}_{ref} = \hat{c}_{p_{ref}}.$$

For compressible flows, the pressure reference is usually defined as

$$\hat{p}_{ref} = \hat{\rho}_{ref}\hat{v}^2_{ref}.$$

Typical reference values are $\hat{\rho}_{ref} = 1.2kg/m^3$ which is approximately the density of air at room temperature or $\hat{v}_{ref}$ as the modulus of a reasonable reference velocity for the problem considered, for example the speed of an airplane. Regarding the reference length, there is no clear way of choosing this. Typically, this is the length of an object that crucially determines the flow, for example the diameter of a cylinder or the length of a plane.

Additionally, we need references for the physical parameters and constants $\hat{\mu}_{ref}$ and $\hat{\kappa}_{ref}$, as well as possibly reference values for the external forces. Reasonable values for air at room temperature and at sea level are $\hat{\mu}_{ref} = 18 \cdot 10^{-6} \frac{kg}{m\,s}$ and $\hat{\kappa}_{ref} = 25 \cdot 10^{-3} \frac{kg\,m}{s^3 K}$. For the nondimensional μ, the Sutherland law gives a relation to the temperature:

$$\mu = T^{\frac{3}{2}} \left(\frac{1 + Su}{T + Su} \right), \tag{2.12}$$

with Su being the Sutherland-constant, which is $Su = \frac{110K}{\hat{T}_{ref}}$ for air.

The Reynolds and Prandtl number, as well as the parameter M are dimensionless quantities, given by:

$$Re = \frac{\hat{\rho}_{ref} \hat{v}_{ref} \hat{x}_{ref}}{\hat{\mu}_{ref}}, \qquad Pr = \frac{\hat{\mu}_{ref} \hat{c}_{p_{ref}}}{\hat{\kappa}_{ref}} \text{ and } M = \frac{\hat{v}_{ref}}{\hat{c}_{ref}}. \tag{2.13}$$

The Reynolds and the Prandtl number determine important flow properties like the size of the boundary layer or if a flow is turbulent. Another important nondimensional quantity is the Mach number Ma, after the german engineer Ernst Mach, which is the quotient of the velocity and the speed of sound $\hat{c}$. The latter is given by

$$\hat{c} = \sqrt{\gamma \frac{\hat{p}}{\hat{\rho}}}. \tag{2.14}$$

For Mach numbers near zero, flow is nearly incompressible. This is called the low Mach number regime. The nondimensional number M is related to the Mach number Ma via $M = \sqrt{\gamma} Ma$ and thus $M = \mathcal{O}_S(Ma)$.

All in all, in the three dimensional case, we obtain the following set of equations for the nondimensional Navier-Stokes equations:

$$\begin{aligned}
\partial_t \rho + \nabla \cdot \mathbf{m} &= 0, \\
\partial_t m_i + \sum_{j=1}^{3} \partial_{x_j} (m_i v_j + p \delta_{ij}) &= \frac{1}{Re} \sum_{j=1}^{2} \partial_{x_j} S_{ij} \qquad i = 1, 2, 3 \\
\partial_t \rho E + \nabla \cdot (H \mathbf{m}) &= \frac{1}{Re} \sum_{j=1}^{2} \partial_{x_j} \left(\sum_{i=1}^{2} S_{ij} v_i - \frac{W_j}{Pr} \right).
\end{aligned} \tag{2.15}$$

In short, using the vector of conservative variables $\mathbf{u} = (\rho, m_1, m_2, m_3, \rho E)^T$, the convective fluxes

$$\mathbf{f}_1^c(\mathbf{u}) = \begin{pmatrix} m_1 \\ m_1v_1 + p \\ m_1v_2 \\ m_1v_3 \\ \rho H v_1 \end{pmatrix}, \quad \mathbf{f}_2^c(\mathbf{u}) = \begin{pmatrix} m_2 \\ m_2v_1 \\ m_2v_2 + p \\ m_2v_3 \\ \rho H v_2 \end{pmatrix}, \quad \mathbf{f}_3^c(\mathbf{u}) = \begin{pmatrix} m_3 \\ m_3v_1 \\ m_3v_2 \\ m_3v_3 + p \\ \rho H v_3 \end{pmatrix}, \tag{2.16}$$

and the viscous fluxes

$$\mathbf{f}_1^v(\mathbf{u}) = \frac{1}{Re}\begin{pmatrix} 0 \\ S_{11} \\ S_{12} \\ S_{13} \\ \mathbf{S}_1 \cdot \mathbf{v} - \frac{W_1}{Pr} \end{pmatrix}, \quad \mathbf{f}_2^v(\mathbf{u}) = \frac{1}{Re}\begin{pmatrix} 0 \\ S_{21} \\ S_{22} \\ S_{23} \\ \mathbf{S}_2 \cdot \mathbf{v} - \frac{W_2}{Pr} \end{pmatrix} \quad \mathbf{f}_3^v(\mathbf{u}) = \frac{1}{Re}\begin{pmatrix} 0 \\ S_{31} \\ S_{32} \\ S_{33} \\ \mathbf{S}_3 \cdot \mathbf{v} - \frac{W_3}{Pr} \end{pmatrix} \tag{2.17}$$

these equations can be written as

$$\mathbf{u}_t + \partial_{x_1}\mathbf{f}_1^c(\mathbf{u}) + \partial_{x_2}\mathbf{f}_2^c(\mathbf{u}) + \partial_{x_3}\mathbf{f}_3^c(\mathbf{u}) = \partial_{x_1}\mathbf{f}_1^v(\mathbf{u}, \nabla\mathbf{u}) + \partial_{x_2}\mathbf{f}_2^v(\mathbf{u}, \nabla\mathbf{u}) + \partial_{x_3}\mathbf{f}_3^v(\mathbf{u}, \nabla\mathbf{u}), \tag{2.18}$$

or in a more compact form:

$$\mathbf{u}_t + \nabla \cdot \mathbf{f}^c(\mathbf{u}) = \nabla \cdot \mathbf{f}^v(\mathbf{u}, \nabla\mathbf{u}). \tag{2.19}$$

Finally, we can write this in integral form as

$$\partial_{\hat{t}} \int_\Omega \mathbf{u} d\Omega + \int_{\partial\Omega} \mathbf{f}^c(\mathbf{u}) \cdot \mathbf{n} ds = \int_{\partial\Omega} \mathbf{f}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n} ds. \tag{2.20}$$

2.3 Source terms

If external forces or source terms are present, these will be modeled by additional terms on the appropriate right hand sides:

$$\partial_{\hat{t}} \int_\Omega \mathbf{u} d\Omega + \int_{\partial\Omega} \mathbf{f}^c(\mathbf{u}) \cdot \mathbf{n} ds = \int_{\partial\Omega} \mathbf{f}^v(\mathbf{u}, \nabla\mathbf{u}) \cdot \mathbf{n} ds + \int_\Omega \mathbf{g} d\Omega.$$

An important example is gravity, which appears as a vector valued source in the momentum equation:

$$\mathbf{g} = (0, 0, 0, g, 0)^T.$$

Another example would be a local heat source in the energy equation or the coriolis force. Additional nondimensional quantities appear in front of the source terms. For the gravitational source term, this is the Froude number

$$Fr = \frac{\hat{v}_{ref}}{\sqrt{\hat{x}_{ref}\hat{g}_{ref}}}. \tag{2.21}$$

2.4 Simplifications of the Navier-Stokes equations

The compressible Navier-Stokes equations are a very general model for compressible flow. Under certain assumptions, simpler models can be derived. The most important simplification of the Navier-Stokes equations are the incompressible Navier-Stokes equations modeling incompressible fluids. In fact, this model is so widely used, that sometimes these equations are referred to as the Navier-Stokes equations and the more general model as the compressible Navier-Stokes equations. They are obtained when assuming that density is constant along the trajectories of particles. In this way, the energy equation is no longer needed and the continuity equation simplifies to the so called divergence constraint:

$$\begin{aligned} \nabla \cdot \mathbf{v} &= 0, \\ \mathbf{v}_t + \mathbf{v} \cdot \nabla \mathbf{v} + \frac{1}{\rho} \nabla p &= \frac{\mu}{\rho} \Delta \mathbf{v}. \end{aligned} \tag{2.22}$$

Note that the density is not necessarily constant here. However, since this is a common modeling assumption, for example for water, the above equations are often called the incompressible Navier-Stokes equations with variable density, whereas the above equations with the density hidden via a redefinition of the other terms are referred to as the incompressible Navier-Stokes equations.

There is a different way of deriving the incompressible Navier-Stokes equations with variable density, namely by looking at the limit $M \to 0$. Using formal asymptotic analysis, it can be shown that the compressible Navier-Stokes equations result in these in the limit.

2.5 The Euler Equations

An important simplification of the Navier-Stokes equations is obtained, if the second order terms (viscosity and heat conduction) are neglected or otherwise put, if the limit Reynolds number to infinity is considered. The resulting set of first order partial differential equations are the so called Euler equations:

$$\begin{aligned} \partial_t \hat{\rho} + \nabla \cdot \hat{\mathbf{m}} &= 0, \\ \partial_t \hat{m}_i + \sum_{j=1}^{d} \partial_{x_j} (\hat{m}_i \hat{v}_j + \hat{p} \delta_{ij}) &= 0, \qquad i = 1, ..., d \\ \partial_t (\hat{\rho} \hat{E}) + \nabla \cdot (\hat{H} \hat{\mathbf{m}}) &= 0. \end{aligned} \tag{2.23}$$

Using the standard nondimensionalization, we obtain the following form of the Euler equations:

$$\begin{aligned}\partial_t \rho + \nabla \cdot \mathbf{m} &= 0,\\ \partial_t m_i + \sum_{j=1}^{d} \partial_{x_j}(m_i v_j + p\delta_{ij}) &= 0, \qquad i = 1, ..., d\\ \partial_t \rho E + \nabla \cdot (H\mathbf{m}) &= 0.\end{aligned} \tag{2.24}$$

As can be seen, no dimensionless reference numbers appear. With the vector of conservative variables $\mathbf{u} = (\rho, m_1, m_2, m_3, \rho E)^T$ and the convective fluxes $\mathbf{f}_1^c(\mathbf{u})$, $\mathbf{f}_2^c(\mathbf{u})$, $\mathbf{f}_2^c(\mathbf{u})$ as before, these equations can be written as

$$\mathbf{u}_t + \partial_{x_1}\mathbf{f}_1^c(\mathbf{u}) + \partial_{x_2}\mathbf{f}_2^c(\mathbf{u}) + \partial_{x_3}\mathbf{f}_3^c(\mathbf{u}) = \mathbf{0}$$

or in more compact form

$$\mathbf{u}_t + \nabla \cdot \mathbf{f}^c(\mathbf{u}) = \mathbf{0}. \tag{2.25}$$

2.6 Boundary and Initial Conditions

Initially, at time $\hat{t}_0$, we have to prescribe values for all variables, where it does not matter if we use the conservative variables or any other set, as long as there is a one to one mapping to the conservative variables. We typically use the primitive variables, as these can be measured quite easily, in contrast to e.g. the conservative variables. Further, if we restrict ourselves to a compact set $D \in \mathcal{U}$, we have to prescribe conditions for the solution on the boundary. This is necessary for numerical calculations and therefore, D is also called the computational domain. The number of boundary conditions needed depends on the type of the boundary.

Initial Conditions At time $t = t_0$, we define a velocity $\mathbf{v}_0$, a density ρ_0 and a pressure p_0. All other values like the energy and the momentum will be computed from these.

Fixed wall Conditions At the wall, no-slip conditions are the conditions to use for viscous flows, thus the velocity should vanish: $\mathbf{v} = 0$. Regarding the temperature, we use either isothermal boundary conditions, where the temperature is prescribed or adiabatic boundary conditions, where the normal heat flux is set to zero.

For the Euler equations, there are fewer possible conditions at the wall, due to the hyperbolic nature. Therefore, the slip condition is employed, thus only the normal velocity should vanish: $v_n = 0$.

Moving walls At a moving wall, the condition is that the flow velocity has to be the same as the wall velocity $\dot{\mathbf{x}}$. This leads to the following equation:

$$\int_\Omega \mathbf{u}_t d\Omega + \int_{\partial\Omega} \mathbf{f}(\mathbf{u}) - \dot{\mathbf{x}} \cdot \mathbf{n} ds = \int_{\partial\Omega} \mathbf{f}^v(\mathbf{u}, \nabla \mathbf{u}) \cdot \mathbf{n} ds. \tag{2.26}$$

Periodic boundaries To test numerical methods, periodic boundary conditions can be useful. Given a set of points $\mathbf{x}_1$ on a boundary Γ_1, these are mapped to a different set of points $\mathbf{x}_2$ on a boundary Γ_2:

$$\mathbf{u}(\mathbf{x}_1) = \mathbf{u}(\mathbf{x}_2). \tag{2.27}$$

Farfield boundaries Finally, there are boundaries that are chosen purely for computational reasons, sometimes referred to as farfield boundaries. From a mathematical point of view, the question is which boundary conditions lead to a well-posed problem and how many boundary conditions can be posed in the first place [116]. For the Euler equations, the answer is that the crucial property is the number of incoming waves, which can be determined using the theory of characteristics. This means that the sign of the eigenvalues (2.30) has to be determined and negative eigenvalues in normal direction correspond to incoming waves. As shown in [192] for the Navier-Stokes equations using the energy method, the number of boundary conditions there differs significantly. Note that the question, if these boundary conditions lead to the same solution on the small domain as for the Navier-Stokes equation on an unbounded domain is open. The number of boundary conditions that lead to a well-posed problem is shown in table 2.1.

	Euler equations	Navier-Stokes equations
Supersonic inflow	5	5
Subsonic inflow	4	5
Supersonic outflow	0	4
Subsonic outflow	1	4

Table 2.1: Number of boundary conditions to be posed in 3D

For the Navier-Stokes equations, a full set of boundary conditions has to be provided at both supersonic and subsonic inflow and one less at the outflow. Intuitively, this can be explained through the continuity equation being a transport equation only, whereas the momentum and energy equations have a second order term. For the Euler equations and subsonic flow at an inflow boundary, we have to specify three values, as we have one outgoing wave corresponding to the eigenvalue $v_n - c < 0$. At the outflow boundary, we have three outgoing waves and one incoming wave, which again corresponds to the eigenvalue $v_n - c$. For supersonic flow, we have only incoming waves at the inflow boundary, respectively no incoming waves, which means that we cannot prescribe anything there.

2.7 Boundary layers

Another important property of the Navier-Stokes equations is the presence of boundary layers [172]. Mathematically, this is due to the parabolic nature and therefore, boundary layers are not present in the Euler equations. We distinguish two types of boundary layers: the velocity boundary layer due to the slip condition, where the tangential velocity changes from zero to the free stream velocity and the thermal boundary layer in case of isothermal boundaries, where the temperature changes from the wall temperature to the free stream temperature. The thickness of the boundary layer is of the order 1/Re, where the velocity boundary layer is a factor of Pr larger than the thermal one. This means that the higher the Reynolds number, the thinner the boundary layer and the steeper the gradients inside.

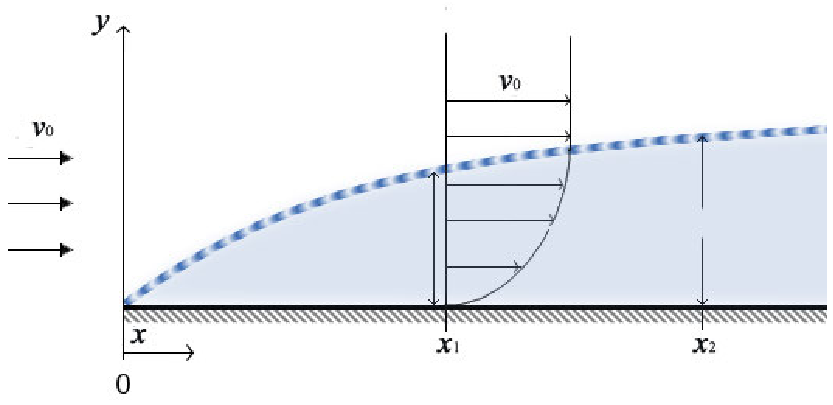


Figure 2.1: Boundary layer; Tuso, CC-by-SA 3.0

An important flow feature that boundary layers can develop is flow separation, where the boundary layer stops being attached to the body, typically by forming a separation bubble.

2.8 Laminar and turbulent flows

When looking at low speed flow around an object, the observation is made that the flow is streamlined and mixing between neighboring flows is very limited. However, when the speed is increased, then at some point the streamlines start to break, eddies appear and neighboring flows mix significantly, see figure 2.2. The first is called the *laminar* flow regime, whereas the other one is the *turbulent* flow regime. In fact, turbulent flows are chaotic in nature. As mentioned in the introduction, we will consider almost only laminar flows.

The same qualitative behavior can be observed dependend on the size of the object and the inverse of the viscosity. More precise, for any object, there is a certain critical

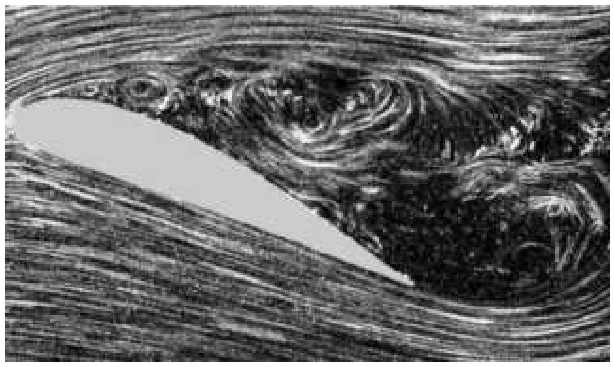


Figure 2.2: Turbulent flow; Jaganath, CC-by-SA 3.0

Reynolds number at which a laminar flow starts to become turbulent. The dependency on the inverse of the viscosity means that air flows are typically turbulent, for example, the Reynolds number of a commercial airliner is between 10^6 and 10^8 for the A-380, whereas the critical Reynolds number is more of the order 10^3. The exact determination of the critical Reynolds number is very difficult.

Simulating and understanding turbulent flows is still a very challenging problem. An important property of turbulent flows is that significant flow features are present at very different scales. The numerical method has to treat these different scales somehow. Essentially there are two strategies to consider in the numerical model: Direct Numerical Simulation (DNS) or turbulence models. DNS uses extremely fine grids to resolve turbulent eddies directly. Unfortunately, resolving the smallest eddies, as shown by Kolmogorov, requires points on a scale of $\text{Re}^{9/4}$ and therefore, this approach is infeasible for practical applications even on modern supercomputers and more importantly, will remain to be so [145]. This leads to the alternative of turbulence models.

2.8.1 Turbulence models

A turbulence model is a model derived from the Navier-Stokes equations that tries to resolve only the larger eddies and not smaller eddies. To this end, the effect of small scale eddies is incorporated using additional terms in the original equations with additional equations to describe the added terms. Examples for these approaches are the Reynolds Averaged Navier-Stokes equations (RANS) and the Large Eddy Simulation (LES).

The RANS equations are obtained by formally averaging the Navier-Stokes equations in a certain sense, an idea that goes back to Reynolds. Thus every quantity is represented by a mean value plus a fluctuation:

$$\phi(x,t) = \overline{\phi}(x,t) + \phi'(x,t).$$

For the definition of the mean value $\overline{\phi}(x,t)$, several possibilities exist, as long as the corresping average of the fluctuation $\overline{\phi'}(x,t)$ is zero. The Reynolds average is used for statistically steady turbulent flows and is given by averaging over a time interval that is significantly smaller than the time step, but large enough to integrate over small eddies

$$\overline{\phi}(x,t) = \frac{1}{T}\int_{-T/2}^{T/2} \phi(x,t+\tau)d\tau,$$

whereas the ensemble average is defined via

$$\overline{\phi}(x,t) = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} v_i.$$

Furthermore, to avoid the computation of mean values of products, the Favre or density weighted average is introduced:

$$\tilde{\phi} = \overline{\rho\phi}/\overline{\rho} \tag{2.28}$$

with a corresponding different fluctuation

$$\phi(x,t) = \tilde{\phi}(x,t) + \phi^{''}(x,t).$$

Due to its hyperbolic nature, the continuity equation is essentially unchanged. In the momentum and energy equations, this averaging process leads to additional terms, namely the so called Reynolds stresses $\mathbf{S}^R$ and the turbulent energy, which have to be described using experimentally validated turbulence models. We thus obtain the RANS equations (sometimes more correctly referred to as the Favre- and Reynolds-Averaged Navier-Stokes equations):

$$\begin{aligned}
\partial_t\overline{\rho} + \nabla\cdot\overline{\rho}\tilde{\mathbf{v}} &= 0,\\
\partial_t\overline{\rho}\tilde{\mathbf{v}} + \sum_{j=1}^{d}\partial x_j(\overline{\rho}\tilde{v}_i\tilde{v}_j) &= -\partial x_j\overline{p}\delta_{ij} + \frac{1}{Re}\sum_{j=1}^{d}\partial x_j\left(\tilde{S}_{ij} + S_{ij}^R\right), \quad i = 1,...,d \qquad (2.29)\\
\partial_t\overline{\rho}\tilde{E} + \nabla\cdot(\overline{\rho}\tilde{H}\tilde{v}_j) &= \sum_{j=1}^{d}\partial x_j\left(\overline{(\frac{1}{Re}S_{ij} - S_{ij}^R)v_i} - \overline{\rho}\widetilde{v_j^{''}} + \widetilde{S_{ij}v_i^{ii}} - \overline{\rho}\widetilde{v_j^{''}k} + \frac{\overline{W}_j}{RePr}\right).
\end{aligned}$$

The Reynolds stresses

$$S_{ij}^R = -\overline{\rho}\widetilde{v_i^{''}v_j^{''}}$$

and the turbulent energy

$$k = \frac{1}{2}\sum_{j=1}^{d}\overline{v_j^{'}v_j^{'}}$$

cannot be related to the unknowns of this equation. Therefore, they have to be described using experimentally validated turbulence models. These differ by the number of additional partial differential equations used to determine the Reynolds stresses and the turbulent energy. There are zero equation models, like the Baldwin-Lomax-model where just an algebraic equation is employed [4], one equation models like the Spallart-Allmaras model [183] or two equation models, as in the well known k-ϵ-models [155].

A more accurate approach is LES, which originally goes back to Smagorinsky, who developed it for meteorological computations [177]. Again, the solution is decomposed into two parts, where this time, a filtering process is used and the unfiltered part (the small eddies) is neglected. However, as for the RANS equations, additional terms appear in the equations, which are called subgrid scale stresses (SGS). These correspond to the effect of the neglected terms and have to be computed using a subgrid scale mode.

Finally, there is the detached eddy simulation (DES) of Spallart [184]. This is a mixture between RANS and LES, where a RANS model is used in the boundary layer, whereas an LES model is used for regions with flow separation. Since then, several improved variants have been suggested, for example DDES.

2.9 Analysis of viscous flow equations

Basic mathematical questions about an equation are: Is there a solution, is it unique and is it stable in some sense? In the case of the Navier-Stokes equations, there are no satisfactory answers to these questions. The existing results provide roughly speaking either long time results for very strict conditions on the initial data or short time results for weak conditions on the initial data. For a review we refer to Lions [125].

2.9.1 Analysis of the Euler equations

As for the Navier-Stokes equations, the analysis of the Euler equations is extremely difficult and the existing results are lacking [36]. First of all, the Euler equations are purely hyperbolic, meaning that the eigenvalues of $\nabla \mathbf{f}^c \cdot \mathbf{n}$ are all real for any $\mathbf{n}$. In particular, they are given by

$$\begin{aligned} \lambda_{1/2} &= |v_n| \pm c, \\ \lambda_{3,4,5} &= |v_n|. \end{aligned} \tag{2.30}$$

Thus, the equations have all the properties of hyperbolic equations. In particular, the solution is monotone, total variation nonincreasing, l_1-contractive and monotonocity preserving. Furthermore, we know that when starting from nontrivial smooth data, the solution will be discontinuous after a finite time.

From (2.30), it can be seen that in multiple dimensions, one of the eigenvalues of the Euler equations is a multiple eigenvalue, which means that the Euler equations are not strictly hyperbolic. Furthermore, the number of positive and negative eigenvalues depends on the relation between normal velocity and the speed of sound. For $v_n > c$, all eigenvalues are positive, for $v_n < c$, one eigenvalue is negative and furthermore, there are zero eigenvalues in the cases $v_n = c$ and $v_n = 0$. Physically, this means that for $v_n < c$, information is transported in two directions, whereas for $v_n > c$, information is transported in one direction only. Alternatively, this can be formulated in terms of the Mach number. Of particular interest is this for the reference Mach number M, since this tells us how the flow of information in most of the domain looks like. Thus, the regime $M < 1$ is called the subsonic regime, $M > 1$ the supersonic regime and additionally, in the regime M, we typically have the situation that we can have locally subsonic and supersonic flows and this regime is called transonic.

Regarding uniqueness of solutions, it is well known that these are nonunique. However, the solutions typically violate physical laws not explicitely modelled via the equations, in particular the second law of thermodynamics that entropy has to be nonincreasing. If this is used as an additional constraint, entropy solutions can be defined that are unique for the one dimensional case.

2.9.2 Analysis of the Navier-Stokes equations

For a number of special cases, exact solutions have been provided, in particular by Helmholtz, who managed to give results for the case of flow of zero vorticity and Prandtl, who derived equations for the boundary layer that allow to derive exact solutions.

Chapter 3

The Space discretization

As discussed in the introduction, we now seek approximate solutions to the continuous equations, where we will employ the methods of lines approach, meaning that we first discretize in space to transform the equations into ordinary differential equations and then discretize in time. Regarding space discretizations, there is a plethora of methods available. The oldest methods are finite difference schemes that approximate spatial derivatives by finite differences, but these become very complicated for complex geometries or high orders. Furthermore, the straightforward methods have problems to mimic core properties of the exact solution like conservation or nonoscillatory behavior. While in recent years, a number of interesting new methods have been suggested, it remains to be seen whether these methods are competitive, respectively where their niches lies. In the world of elliptic and parabolic problems, finite element discretizations are standard. These use a set of basis functions to represent the approximate solution and then seek the best approximation defined by a Galerkin condition. For elliptic and parabolic problems, these methods are backed by extensive results from functional analysis, making them very powerful. However, they have problems with convection dominated problems in that there, additional stabilization terms are needed. This is an active field of research in particular for the incompressible Navier-Stokes equations, but the use of finite element methods for compressible flows is very limited.

The methods that are standard in industry and academia are finite volume schemes. These use the integral form of a conservation law and consider the change of the mean of a conservative quantity in a cell via fluxes over the boundaries of the cell. Thus, the methods inherently conserve these quantities and furthermore can be made to satisfy additional properties of the exact solutions. Finite volume methods will be discussed in section 3.2. A problem of finite volume schemes is their extension to orders above two. A way of achieving this are discontinous Galerkin methods that use ideas originally developed in the finite element world. These will be considered in section 3.8.

3.1 Structured and unstructured Grids

Before we describe the space discretization, we will discuss different types of grids, namely structured and unstructured grids. The former are grids that have a certain regular structure, whereas unstructured grids do not. In particular, cartesian grids are structured and also so called O- and C-type grids, which are obtained from mapping a cartesian grid using a Möbius transformation. The main advantage of structured grids is that the data structure is simpler, since for example the number of neighbors of a grid cell is a priori known, and thus the algorithm is easier to program. Furthermore, the simpler geometric structure also leads to easier analysis of numerical methods which often translates in more robustness and speed.

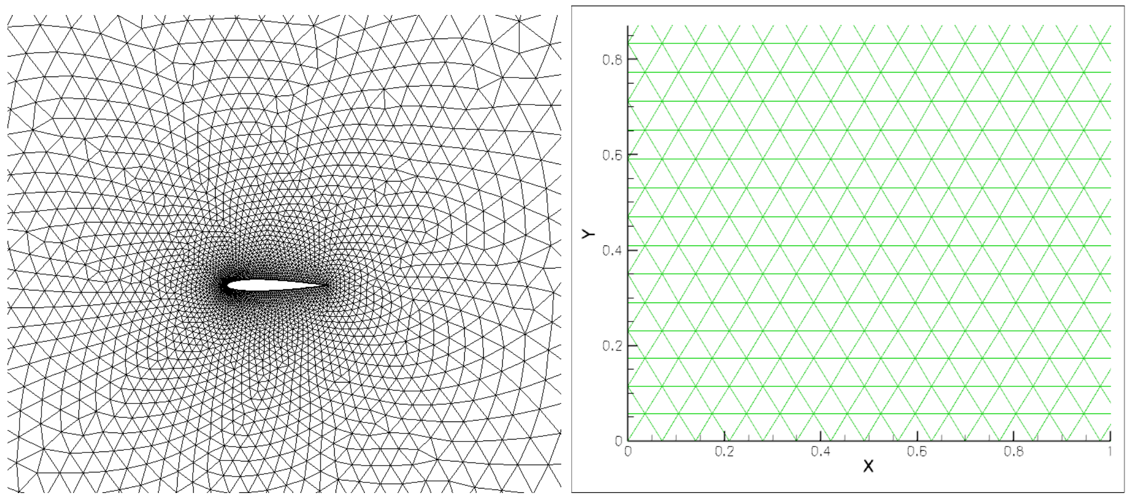


Figure 3.1: Example of unstructured (left) and structured (right) triangular grids

On the other hand, the main advantage of unstructured grids is that they are geometrically much more flexible, allowing for a better resolution of arbitrary geometries.

When generating a grid for the solution of the Navier-Stokes equations, an important feature to consider is the boundary layer. Since there are huge gradients in normal direction, but not in tangential direction, it is useful to use cells in the boundary layer that have a high aspect ratio, the higher the Reynolds number, the higher the aspect ratio. Furthermore, to avoid cells with extreme angles, the boundary layer is often discretized using a structured grid.

Regarding grid generation, this continues to be a bottle neck in CFD calculations in that automatic procedures for doing so are missing. Possible codes are for example the commercial software CENTAUR [35] or the open source Gmsh [67].

3.2 Finite Volume Methods

The equations we are trying to solve are so called conservation, respectively balance laws. For these, finite volume methods are the most natural to use. Basis for those is the integral form of the equation. An obvious advantage of this formulation is that discontinuous solutions of some regularity are admissible. This is favourable for nonlinear hyperbolic equations, because shocks are a common feature of their solutions. We will present the implemented method only briefly and refer the interested reader for more information to the excellent textbooks [69, 93, 94, 123, 124] and the more concise treatises [146, 7]. We start the derivation with the integral form (2.20):

$$\frac{d}{dt}\int_{\Omega} \mathbf{u}(\mathbf{x},t)\; d\Omega + \int_{\partial\Omega} \mathbf{f}(\mathbf{u}(\mathbf{x},t)) \cdot \mathbf{n}\; d\mathbf{s} = \int_{\Omega} \mathbf{g}(\mathbf{x},t,\mathbf{u}(\mathbf{x},t))\; d\Omega. \tag{3.1}$$

Here, Ω is a so called control volume or cell with outer normal unit vector $\mathbf{n}$. The only condition we need to put on Ω for the derivation to work is that it has a Lipschitz continuous boundary. However, we now assume that all control volumes have polygonal boundaries. This is not a severe restriction and allows for a huge amount of flexibility in grid generation, which is another advantage of finite volume schemes. Thus we decompose the computational domain D into a finite number of polygons in 2D, respectively polygonally bounded sets in 3D.

We denote the i'th cell by Ω_i and its volume by $|\Omega_i|$. Edges will be called e with the edge between Ω_i and Ω_j being e_{ij} with length $|e_{ij}|$, whereby we use the same notation for surfaces in 3D. Furthermore, we denote the set of indices of the cells neighboring cell i with $N(i)$. We can therefore rewrite (3.1) for each cell as

$$\frac{d}{dt}\int_{\Omega_i} \mathbf{u}(\mathbf{x},t)\; d\Omega + \sum_{j\in N(i)} \int_{e_{ij}} \mathbf{f}(\mathbf{u}(\mathbf{x},t)) \cdot \mathbf{n}\; d\mathbf{s} = \int_{\Omega_i} \mathbf{g}(\mathbf{x},t,\mathbf{u}(\mathbf{x},t))\; d\Omega. \tag{3.2}$$

The key step towards a numerical method is now to consider the mean value

$$\mathbf{u}_i(t) := \frac{1}{|\Omega_i|}\int_{\Omega_i} \mathbf{u}(\mathbf{x},t) d\Omega$$

of $\mathbf{u}(\mathbf{x},t)$ in every cell Ω_i and to use this to approximate the solution in the cell. Under the condition that Ω_i does not change with time we obtain an evolution equation for the mean value in a cell:

$$\frac{d}{dt}\mathbf{u}_i(t) = -\frac{1}{|\Omega_i|}\sum_{j\in N(i)} \int_{e_{ij}} \mathbf{f}(\mathbf{u}(\mathbf{x},t)) \cdot \mathbf{n}\; d\mathbf{s} + \frac{1}{|\Omega_i|}\int_{\Omega_i} \mathbf{g}(\mathbf{x},t,\mathbf{u}(\mathbf{x},t))\; d\Omega. \tag{3.3}$$

We will now distinguish two types of schemes, namely cell-centered and cell-vertex schemes. For cell-centered schemes, the grid will be used as generated, meaning that the unknown quantities do not correspond to vertices, but to the cells of the grid. This is a very common type of scheme, in particular for the Euler equations. In the case of cell-vertex

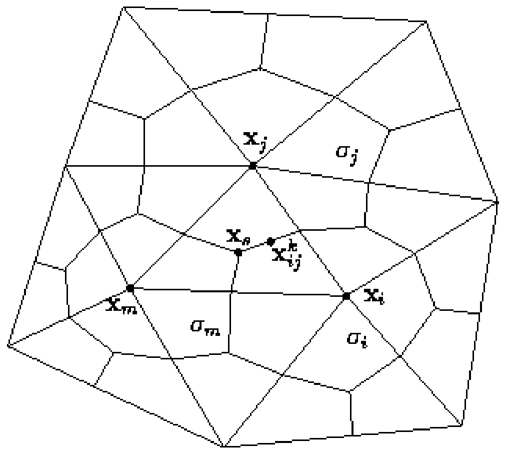


Figure 3.2: Primary triangular grid and corresponding dual grid for a cell-vertex scheme.

schemes, the unknowns are located in the vertices of the original (primary) grid. This is possible after creating a dual grid with dual control volumes, by computing the barycenter of the original grid and connecting this to the midpoints of the edges of the containing primary cell, see fig. 3.2. The finite volume method then acts on the dual cells.

For cartesian grids, there is no real difference between the two approaches, but for shapes like triangles and tetrahedrons, the difference is significant. For example, the angles between two edges are larger, which later implies a smaller discretization error and thus the possibility of using larger cells. However, the main purpose of these schemes is to make the calculation of velocity gradients on unstructured grids easier, which is important in the context of Navier-Stokes equations. This will be discussed in section 3.3.3.

3.3 The Line Integrals and Numerical Flux Functions

In equation (3.2), line integrals of the flux along the edges appear. A numerical method thus needs a mean to compute them. On the edge though, the numerical solution is usually discontinuous, because it consists of the mean values in the cells. Therefore, a numerical flux function is required. This takes states from the left and the right side of the edge and approximates the exact flux based on these states. For now, we assume that the states are the mean values in the respective cells, but later when considering higher order methods in section 3.7, other values are possible. The straight forward way to define a numerical flux function would be to simply use the average of the physical fluxes from the left and the right. However, this leads to an unconditionally unstable scheme and therefore, additional stabilizing terms are needed.

A numerical flux function $\mathbf{f}^N(\mathbf{u}_L, \mathbf{u}_R, \mathbf{n})$ is called consistent, if it is Lipschitz continuous in the first two arguments and if $\mathbf{f}^N(\mathbf{u}, \mathbf{u}, \mathbf{n}) = \mathbf{f}(\mathbf{u}, \mathbf{n})$. This essentially means that if we refine the discretization, the numerical flux function will better approximate the physical flux.

We furthermore call a numerical flux function $\mathbf{f}^N$ *rotationally invariant* if a rotation of the coordinates does not change the flux. The physical flux has this property, therefore it is reasonable to require this also of the numerical flux. More precisely, this property is given by

$$\mathbf{T}^{-1}(\mathbf{n})\mathbf{f}^N(\mathbf{T}(\mathbf{n})\mathbf{u}_L, \mathbf{T}(\mathbf{n})\mathbf{u}_R; (1,0)^T) = \mathbf{f}^N(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}).$$

In two dimensions, the matrix $\mathbf{T}$ is given via

$$\mathbf{T}(\mathbf{n}) = \begin{pmatrix} 1 & & & \\ & n_1 & n_2 & \\ & -n_2 & n_1 & \\ & & & 1 \end{pmatrix}.$$

The rotation matrix in three dimensions is significantly more complicated. This property can be made use of in the code, as it allows to assume that the input of the numerical flux function is aligned in normal direction and therefore, it is sufficient to define the numerical flux functions only for $\mathbf{n} = (1,0,0)^T$.

In the derivation so far, we have not used any properties of the specific equation, except for the rotational invariance, which is a property that most physical systems have. This means that the flux functions contain the important information about the physics of the equations considered and that for finite volume methods, special care should be taken in designing the numerical flux functions. There is a significant advantage to this, namely that given a finite volume code to solve a certain equation, it is rather straightforward to adjust the code to solve a different conservation law.

Employing any flux function, we can now approximate the line integrals by a quadrature formula. A gaussian quadrature rule with one Gauss point in the middle of the edge already achieves second order accuracy, which is sufficient for the finite volume scheme used here. Thus, we obtain a semidiscrete form of the conservation law, namely a finite dimensional nonlinear system of ordinary differential equations. This way of treating a partial differential equation is also called the method of lines approach. For each single cell, this differential equation can be written as:

$$\frac{d}{dt}\mathbf{u}_i(t) + \frac{1}{|\Omega_i|}\sum_{j\in N(i)} |e_{ij}|\mathbf{T}^{-1}(\mathbf{n}_{ij})\mathbf{f}^N(\mathbf{T}(\mathbf{n}_{ij})\mathbf{u}_i, \mathbf{T}(\mathbf{n}_{ij})\mathbf{u}_j; (1,0,0)^T) = \mathbf{0}, \tag{3.4}$$

where the input of the flux function are the states on the left hand, respectively right hand side of the edge.

3.3.1 Discretization of the inviscid fluxes

We will now briefly describe the inviscid flux functions employed here, for more detailed information consult [94, 198] and the references therein. Van Leers flux vector splitting [207] is not a good method to compute flows with strong boundary layers. Another alternative is the Rusanov flux. We further mention the CUSP and the SLIP scheme developed by Jameson. These use an artificial diffusion term. Finally, there is the famous Roe scheme. In the context of the later explained DG methods, the extremely simple Rusanov flux is also employed. In the descriptions of the flux functions, we will assume that they have the vectors $\mathbf{u}_L$ and $\mathbf{u}_R$ as input, with their components indexed by L and R, respectively.

HLLC

The HLLC flux of Toro is from the class of approximate Riemann solvers that consider a Riemann problem and solve that approximately. HLLC tries to capture all waves in the problem to then integrate an approximated linearized problem, see figure 3.3. It is an

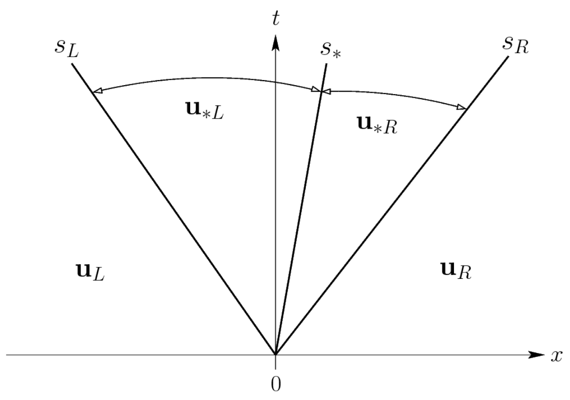


Figure 3.3: The solution of the linearized Riemann problem used in HLLC: Four different states $\mathbf{u}_L$, $\mathbf{u}_R$, $\mathbf{u}_{*L}$ and $\mathbf{u}_{*R}$ are separated by two shock waves with speeds s_L and s_R and a contact discontinuity of speed s_*

extension of the HLL Riemann solver of Harten, Hyman and van Leer, but additionally includes a contact discontinuity, thus the C. To this end, the speeds s_L and s_R of the left and right going shock are determined and the speed of the contact discontinuity is approximated as s^*. With these, the flux is determined as

$$\mathbf{f}(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}) = \begin{cases} \mathbf{f}(\mathbf{u}_L), & 0 \leq s_L, \\ \mathbf{f}_L^*, & s_L \leq 0 \leq s^*, \\ \mathbf{f}_R^*, & s^* \leq 0 \leq s_R, \\ \mathbf{f}(\mathbf{u}_R), & s_R \geq 0, \end{cases} \tag{3.5}$$

where

$$\mathbf{f}_L^* = \mathbf{f}(\mathbf{u}_L) + s_L(\mathbf{u}_{*L} - \mathbf{u}_L)$$

and analogously,

$$\mathbf{f}^*_R = \mathbf{f}(\mathbf{u}_R) + s_R(\mathbf{u}_{*R} - \mathbf{u}_R).$$

Here, the intermediate states $\mathbf{u}_{*K}$ are given by

$$\mathbf{u}_{*K} = \rho_K \left(\frac{s_K - v_{1_K}}{s_K - s^*} \right) \begin{pmatrix} 1 \\ s^* \\ v_{2_K} \\ v_{3_K} \\ \frac{E_K}{\rho_K} + (s^* - v_{1_K})(s^* - \frac{p_K}{\rho_K(s_K - v_{1_K})}) \end{pmatrix}.$$

The intermediate speed is given by

$$s^* = \frac{1}{2}\{v_1\} + \frac{1}{2}(p_L - p_R)/(\hat{\rho}\hat{c}),$$

where $\{v_1\}$ is the left-right averaged quantity and

$$\hat{\rho} = \frac{1}{2}\{\rho\}, \quad \hat{c} = \frac{1}{2}\{c\}.$$

Furthermore, the speeds of the left and right going shock are found as

$$s_L = v_{1_L} - c_L q_L, \quad s_R = v_{1_R} + c_R q_R,$$

Finally, the wave speeds q_K for $K = L, R$ are obtained from

$$q_k = \begin{cases} 1, & p^* \leq p_K \\ \sqrt{1 + \frac{\gamma+1}{\gamma}(\frac{p^*}{p_K} - 1)}, & p^* > p_K \end{cases}$$

with

$$p^* = \frac{1}{2}\{p\} + \frac{1}{2}(v_{1_L} - v_{1_R})/(\hat{\rho}\hat{c}).$$

AUSMDV

By contrast to many other numerical flux functions like HLLC, the idea behind the fluxes of the AUSM family is not to approximate the Riemann problem and integrate that, but to approximate the flux directly. To this end, the flux is written as the sum of a pressure term and a flow term. The precise definition of these then determines the exact AUSM-type scheme. The AUSMDV flux [217] is actually a combination of several fluxes, namely AUSMD, AUSMV, the flux function of Hänel and a shock fix. First, we have

$$\mathbf{f}^{AUSMD}(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}) = \frac{1}{2}[(\rho v_n)_{1/2}(\Psi_L + \Psi_R) + |(\rho v_n)_{1/2}|(\Psi_L - \Psi_R)] + \mathbf{p}_{1/2}, \tag{3.6}$$

with

$$\Psi = (1, v_1, v_2, v_3, H)^T \text{ and } \mathbf{p}_{1/2} = (0, p_{1/2}, 0, 0, , 0)^T.$$

Here,

$$p_{1/2} = p_L^+ + p_R^-,$$

with

$$p_{L/R}^{\pm} = \begin{cases} \frac{1}{4}p_{L/R}(M_{L/R} \pm 1)^2(2 \mp M_{L/R}) & \text{if } |M_{L/R}| \leq 1, \\ \frac{1}{2}p_{L/R}\frac{v_{n,L/R} \pm v_{n,L/R}}{v_{n,L/R}} & \text{else.} \end{cases}$$

The local Mach numbers are defined as

$$M_{L,R} = \frac{v_{n,L/R}}{c_m},$$

with $c_m = \max\{c_L, c_R\}$. Furthermore, the mass flux is given by

$$(\rho v_n)_{1/2} = v_{n,L}^+ \rho_L + v_{n,R}^- \rho_R,$$

with

$$v_{n,L/R}^{\pm} = \begin{cases} \alpha_{L,R}\left(\pm\frac{(v_{n,L/R} \pm c_m)^2}{4c_m} - \frac{v_{n,L/R} \pm |v_{n,L/R}|}{2}\right) + \frac{v_{n,L/R} \pm |v_{n,L/R}|}{2} & \text{if } |M_{L/R}| \leq 1, \\ \frac{v_{n,L/R} \pm v_{n,L/R}}{2} & \text{else.} \end{cases}$$

The two factors α_L and α_R are defined as

$$\alpha_L = \frac{2(p/\rho)_L}{(p/\rho)_L + (p/\rho)_R} \text{ and } \alpha_R = \frac{2(p/\rho)_R}{(p/\rho)_L + (p/\rho)_R}.$$

As the second ingredient of AUSMDV, the AUSMV flux is identical to the AUSMD flux, except that the first momentum component is changed to

$$f_{m_1}^{AUSMV}(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}) = v_{n,L}^+(\rho v_n)_L + v_{n,R}^-(\rho v_n)_R + p_{1/2} \tag{3.7}$$

with $v_{n,L}^+$ and $v_{n,R}^-$ defined as above.

The point is that AUSMD and AUSMV have different behavior at shocks in that AUSMD causes oscillations at these. On the other hand, AUSMV produces wiggles in the velocity field at contact discontinuities. Therefore, the two are combined using a gradient based sensor s:

$$\mathbf{f}^{AUSMD+V} = \left(\frac{1}{2} + s\right)\mathbf{f}^{AUSMD}(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}) + \left(\frac{1}{2} - s\right)\mathbf{f}^{AUSMV}(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}). \tag{3.8}$$

The sensor s depends on the pressure gradient in that

$$s = \frac{1}{2}\min\left\{1, K\frac{|p_R - p_L|}{\min\{p_R, p_R\}}\right\},$$

where we chose $K = 10$.

Now, the AUSMD+V still has problems at shocks, which is why the Hänel flux is used to increase damping. This is given by

$$\mathbf{f}^{Haenel}(\mathbf{u}_L, \mathbf{u}_R; \mathbf{n}) = v_{n,L}^{+}\rho_L \mathbf{\Psi}_L + v_{n,R}^{-}\rho_R \mathbf{\Psi}_R + \mathbf{p}_{1/2}, \tag{3.9}$$

where everything is defined as above, except that $\alpha_L = \alpha_R = 1$ and c_m is replaced by c_L, respectively c_R.

Furthermore, a damping function $\mathbf{f}^D$ is used at the sonic point to make the flux function continuous there. Using the detectors

$$A = ((v_{n,L} - c_L) < 0) \,\&\, (v_{n,R} - c_R) > 0)$$

and

$$B = ((v_{n,L} + c_L) < 0) \,\&\, (v_{n,R} + c_R) > 0),$$

as well as the constant $C = 0.125$, we define the damping function as

$$\mathbf{f}^D = \begin{cases} C((v_{n,L} - c_L) - (v_{n,R} - c_R))((\rho\Phi)_L) - (\rho\Phi)_R), & \text{if } A \,\&\, \neg B, \\ C((v_{n,L} + c_L) - (v_{n,R} + c_R))((\rho\Phi)_L) - (\rho\Phi)_R), & \text{if } \neg A \,\&\, B, \\ \mathbf{0}, & \text{else.} \end{cases} \tag{3.10}$$

Finally, we obtain

$$\mathbf{f}^{AUSMDV} = (1 - \delta_{2,S_L+S_R})(\mathbf{f}^D + \mathbf{f}^{AUSMD+V}) + (\delta_{2,S_L+S_R})\mathbf{f}^{Haenel}, \tag{3.11}$$

where

$$S_K = \begin{cases} 1, & \text{if } \exists j \in N(i) \text{ with } (v_{n_i} - c_i) > 0 \,\&\, (v_{n_j} - c_j) < 0 \\ & \qquad \text{or } (v_{n_i} + c_i) > 0 \,\&\, (v_{n_j} + c_j) < 0, \\ 0, & \text{else,} \end{cases}$$

is again a detector for the sonic point near shocks.

3.3.2 Low Mach numbers

In the case of low Mach numbers, meaning $M < 0.1$, care has to be taken, since all standard flux functions will produce wrong solutions on reasonable grids, in particular in the pressure field. In figure 3.4 (left), the result of a steady state computation is shown using a standard Riemann solver. What happens here is that the spatial fluctuations in the pressure are of the order $\mathcal{O}(M)$, instead of $\mathcal{O}(M^2)$ as would be expected from an asymptotic analysis of the continuous equations [79]. From a historical point of view, the reason is that for decades, the most important applications in CFD were hypersonic and supersonic flows and thus, the standard methods were designed for the resolution of strong shocks. The mathematical reason are jumps in the discrete normal velocity, as was shown by analyzing Riemann problems in [78]. Furthermore, as shown in [167], this problem does not exist on triangular grids.

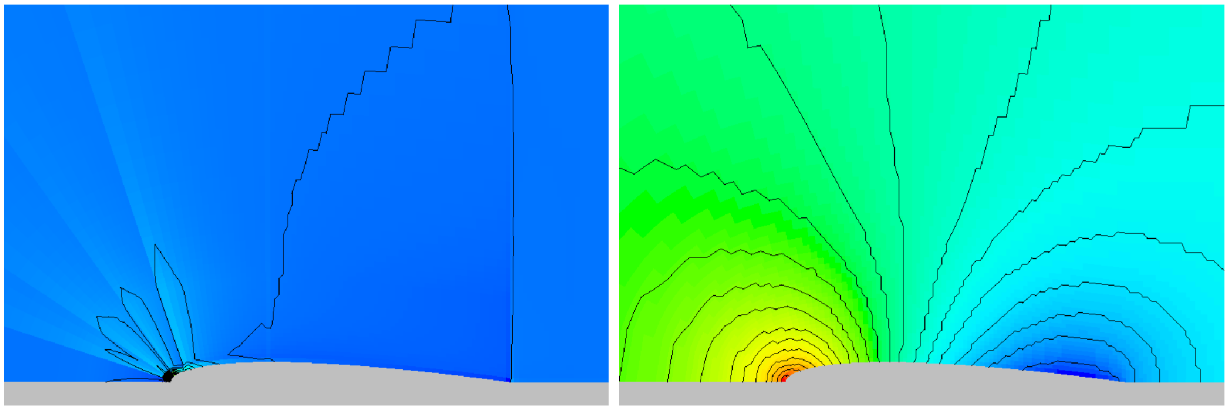


Figure 3.4: Pressure isolines of a NACA0012 simulation at Mach 0.001 using a standard flow solver (left) and a low Mach preconditioned solver (right).

One solution is the preconditioning technique of Guillard and Viozat [79], as demonstrated in figure 3.4 (right). However, this turns out to be unacceptably inefficient if combined with explicit time integration, as was shown by von Neumann stability analysis in [27] and later for a more general case in [48]. There, it is shown that the time step size has to decrease asymptotically with the Mach number squared. Another suggestion is the AUSM+up flux of Liou [127], which also solves the accuracy problem. However, this method has a number of parameters which are difficult to tune and, although stable, shows a similar dependency of the solution on the time step as the preconditioning technique of Guillard and Viozat.

Therefore, more recent fixes focus on the decrease of the jumps in the normal component of the velocity at cell interfaces. Since these jumps are a result of the discretization, they can be mitigated by using higher order methods. For example, Thornber et. al. use a specific limiter that is 5th order on cartesian grids in combination with an averaging of the normal velocity at the cell interfaces to obtain good results at low Mach numbers, while showing that this approach does not work for arbitrary limiters [196]. Unfortunately, their approach is restricted to cartesian grids. A more promising approach for unstructured grids is the following of Rieper [166]. He suggests the method LMRoe, a modification of Roe's flux function, where the jump Δv_n in the normal component across the interface is replaced with

$$\Delta v_n = \min(1, \tilde{M})\Delta v_n$$

and the local Mach number $\tilde{M}$ is given by

$$\tilde{M} = \frac{|v_{n_{ij}}| + |v_{t_{ij}}|}{c_{ij}}$$

and all quantities are supposed to be Roe averaged velocities, which means that we consider

$$\phi_{ij} = \frac{\sqrt{\rho_i}\phi_i + \sqrt{\rho_j}\phi_j}{\sqrt{\rho_i} + \sqrt{\rho_j}}.$$

For this fix, Rieper proves by asymptotic analysis that the resulting method leads to correctly scaled pressure and by von Neumann stability analysis that the time step scales with $\mathcal{O}(M)$.

3.3.3 Discretization of the viscous fluxes

The viscous fluxes (2.17) are easier to discretize, because, due to the parabolic nature of the second order terms, central differences do not lead to an unconditionally unstable scheme. It is therefore possible to base the evaluation of the viscous fluxes at an edge on simple averages over the neighboring cells:

$$\phi_{ij} = \frac{\phi_i + \phi_j}{2}.$$

Here, ϕ corresponds to one of the needed quantities in the viscous fluxes, namely the viscosity ν, a velocity component v_i or the thermal conductivity κ.

Furthermore, gradients of velocity and temperature need to be computed on the edge. Here, the naive implementation using arithmetic means unfortunately leads to a decoupling of neighboring cells, leading to the so called checker board effect. Therefore, the coupling between cells needs to be recovered in some way.

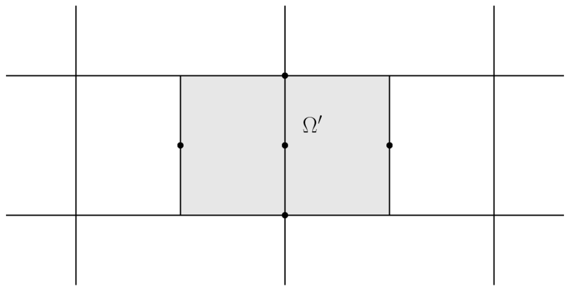


Figure 3.5: The dual cell Ω' used to compute the gradients for a structured cell-centered method.

In the context of a structured cell-centered finite-volume method, this is done by introducing a dual cell Ω' around the edge point where we want to compute the derivative (see figure 3.5). Now we assume that a higher order finite volume method is used, as explained in the later section 3.7.2. This means that the values of the derivatives at the cell centers are given. The others are obtained using arithmetic means over all neighboring cells. The derivative of a quantity ϕ at the edge point is then approximated using Green's theorem:

$$\int_\Omega \frac{\partial \phi}{\partial x} d\Omega = \int_{\partial\Omega} \phi n ds.$$

Thus, we obtain

$$\frac{\partial \phi}{\partial x} = \frac{1}{|\Omega'|} \int_{\partial\Omega'} \phi n ds \approx \frac{1}{\Omega'} \sum_{e_m \in \partial\Omega'} \phi_m |e_m n|. \tag{3.12}$$

As an example in 2D, we have

$$\frac{\partial v_1}{\partial x_1} \approx \frac{1}{\Delta x \Delta y}((v_1)_1 - (v_1)_2)\Delta y.$$

The generalization of this approach to unstructured grids leads to a rather complicated scheme, but there are two possible remedies. For cell-centered methods, the following heuristic approach is commonly used [31]. First, compute intermediate approximations of the gradients using arithmetics means:

$$\nabla \phi_i = \frac{1}{|\Omega_i|} \sum_{e_{ij} \in \partial\Omega_i} \frac{1}{2}(\phi_i + \phi_j)\mathbf{n}_{ij}|e_{ij}|.$$

Then, the means of these are taken:

$$\bar{\nabla}\phi_{ij} = \frac{1}{2}(\nabla\phi_i + \nabla\phi_j).$$

Finally, using the approximated directional derivative

$$\left(\frac{\partial \phi}{\partial l}\right)_{ij} := \frac{\phi_j - \phi_i}{|e_{ij}|}$$

and the normalized vector $\mathbf{r}_{ij}$, pointing from cell center i to cell center j, we obtain the final corrected approximation

$$\nabla\phi_{ij} = \bar{\nabla}\phi_{ij} - \left(\bar{\nabla}\phi_{ij} \cdot \mathbf{r}_{ij} - \left(\frac{\partial \phi}{\partial l}\right)_{ij}\right)\mathbf{r}_{ij}. \tag{3.13}$$

Alternatively, if a structured cell-vertex method is employed, we can simply use (3.12) to obtain the derivative, where as dual cell Ω', we just use the original primal cell. For an unstructured cell-vertex method using triangles and tetrahedrons as primary cells, we can define a unique gradient on these based on the unique linear function interpolating the values in the nodes of the primary grid [142]. In the threedimensional case we have four points $\mathbf{x}_1, ..., \mathbf{x}_4$ with the values there called $\phi_1, ..., \phi_4$. We then obtain for the components of the gradient

$$\nabla\phi_i = \frac{\det \mathbf{A}_i}{\det \mathbf{A}}, \tag{3.14}$$

where

$$\mathbf{A} = \begin{pmatrix} (\mathbf{x}_4 - \mathbf{x}_1)^T \\ (\mathbf{x}_2 - \mathbf{x}_1)^T \\ (\mathbf{x}_3 - \mathbf{x}_1)^T \end{pmatrix}$$

and $\mathbf{A}_i$ is obtained by replacing the i'th column of $\mathbf{A}$ with the vector $((\phi_2 - \phi_1), (\phi_3 - \phi_1), (\phi_4 - \phi_1))^T$. In the two dimensional case, the formulas are accordingly derived from Cramer's rule.

3.4 Convergence theory for finite volume methods

3.4.1 Hyperbolic conservation laws

By contrast to finite element methods for elliptic equations, the convergence theory for numerical methods for conservation laws is unsatisfying, in particular for the relevant case of nonlinear systems. Nevertheless, during the last decade, significant advancements have been made, so that the theory for scalar one dimensional equations has reached a certain maturity. An important property of finite volume schemes is that they are conservative, meaning that the total amount of a conservative quantity inside the computational domain is changed only through terms on the boundary and source terms. The theorem of Lax-Wendroff tells us that this is a good thing: If a scheme is conservative and consistent and the numerical solution is of bounded variation, then it converges to a weak solution of the conservation law, if it converges.

Here, the total variation of a function in space is defined as

$$TV(u(x)) = \int |u'(x)| dx. \tag{3.15}$$

In the case of a grid function, this simplifies to

$$TV(u) = \sum_i |u_i - u_{i-1}|. \tag{3.16}$$

Convergence proofs become possible using this concept, since a set of functions with a bounded total variation and a compact support is compact in L_1. The space of functions in $\mathbb{R}^d$ with bounded total variation is called $BV(\mathbb{R}^d)$.

Furthermore, a scheme is called monotone, if it has the following property: Given two numerical solutions u^n and v^n with $u^n \geq v^n$, it holds that $u^{n+1} \geq v^{n+1}$.

We thus have the following theorem [117], which says that for a scalar equation, a monotone scheme converges to the weak entropy solution of the equations and gives an a priori error estimate:

Theorem 2 *Let $u_0(x)$ be in $BV(\mathbb{R}^d)$ and $u(x,t)$ be a weak entropy solution of the conservation law. Furthermore, let u_h be a numerical solution obtained by a monotone scheme with explicit Euler integration in time with a suitably restricted time step. Let K be a specific subset of $\mathbb{R}^d \times \mathbb{R}^+$. Then there exists a constant $C > 0$, such that*

$$\|u - u_h\|_{L_1(K)} \leq Ch^{1/4}.$$

In the one dimensional case, the statement goes with $h^{1/2}$ and this convergence rate is optimal.

The time step restriction in this theorem will be discussed further in the next chapter in section 4.2.

When we consider a one dimensional linear system instead of a scalar equation, it is possible to apply the results to the diagonalized form of the equations and thus, they carry through. However, while diagonalization is still possible for nonlinear systems, it depends on the solution and thus, the equations do not truly decouple. Therefore, the convergence theory for nonlinear systems is very limited.

3.4.2 Parabolic conservation laws

In the case of parabolic conservation laws, there are even fewer results than for hyperbolic equations. This is counterintuitive, since a major problem for the analysis of hyperbolic conservation laws are the discontinuities, which are smoothed through viscous terms. However, the additional terms lead to other difficulties. For example, the total variation is not guaranteed to be nonincreasing for the exact solutions. There is extensive analysis of the linear convection diffusion equation. A more interesting result for nonlinear scalar conservation laws with a diffusion term in multiple dimensions is due to Ohlberger [150]. Since the result is very technical, we give only the gist of it. However, we will discuss the implied restriction on the time step for reasons of stability in section 4.2.

Theorem 3 *Consider the equation in $\mathbb{R}^d$*

$$u_t + \nabla \cdot f(u) = \epsilon \Delta u$$

with $1 \gg \epsilon > 0$ and reasonably regular initial data with compact support and a reasonably regular function f. Assume a finite volume discretization on a reasonable grid and a time step Δt that is small in a certain sense. Then there is a constant $K > 0$, such that

$$\|u - u_h\|_{L_1(\mathbb{R}^d \times \mathbb{R}^+)} \leq K(\epsilon + \sqrt{\Delta x} + \sqrt{\Delta t})^{1/2} \tag{3.17}$$

As in the case of hyperbolic equations, there are no results for the case of nonlinear systems in multiple dimensions, in particular not for the Navier-Stokes equations.

3.5 Boundary Conditions

If an edge is part of the boundary of the computational domain, the numerical flux cannot be defined as before and we have to take a different approach on the edge. With respect to implementation, there are two ways of doing this. The first is to define a flux corresponding to the boundary condition on the edge. Alternatively, a layer of ghost cells is defined. In these, values are prescribed, such that when the numerical flux is computed based on these values, a certain boundary condition is realized. The precise condition depends on the type of boundary.

3.5.1 Fixed wall

At a fixed wall in the case of the Euler equations, we use slip conditions as in section 2.6 for the continuous equations, meaning there should be no flux through the boundary, but tangential to the boundary, no conditions are imposed. Therefore, at the evaluation points on the wall the condition $\mathbf{v} \cdot \mathbf{n} = 0$ has to hold. For the Navier-Stokes equations, we have to use no-slip boundary conditions thus we require the solution to have zero velocity in the boundary points: $\mathbf{v} = \mathbf{0}$. In addition, a boundary condition for the heat flux term has to be given. This is again done corresponding to the continuous equations as either isothermal or adiabatic, thus either prescribing a value for temperature or for the heat flux.

For the slip conditions using ghost cells, we have to prescribe a negative normal velocity, such that this adds up to zero on the wall, all other values are chosen the same as in the neighboring cell. Using boundary fluxes, the prescribed flux is $\mathbf{f} = (0, n_1 p, ..., , n_d p, 0)^T$. Regarding no-slip conditions, the convective boundary flux is the same and there is no additional contribution from the viscous stress tensor, since the velocity is zero. In a ghost cell implementation, this would be realized by setting all velocity components to the appropriate negative value.

In the case of an adiabatic wall condition, the additional flux term on top of the no-slip flux would be zero and similarly, nothing needed to be changed about the ghost cells. For an isothermal wall, the temperature values are extrapolated in the ghost cell implementation, whereas for boundary fluxes, a corresponding temperature gradient is determined.

Note that the slip condition can be used in the case of symmetric flows to cut the computational domain in half.

3.5.2 Inflow and outflow boundaries

Other boundaries are artificial ones where the computational domain ends, but not the physical one. Thus, a boundary flux has to be found that leads to the correct solution in the limit of mesh width to zero or that otherwise spoken, makes the artificial boundary interfer as little as possible with the solution on the larger domain.

Several types of boundary conditions are possible and should be used depending on the problem to solve. The most simple ones are *constant interpolation boundary conditions* which means that we use Neumann boundary conditions where we set the derivative on the edge to zero. Using ghost cells, this corresponds to copying the value from the interior cell to the ghost cell. However, this reduces the order at the boundary to one and leads to problems when significant tangential flows are present.

Another option are *far field boundary conditions*, where the value in the ghost cell is always set to the initial value. This type of boundary conditions is particularly useful for the computation of steady flows. However, it can happen that in this way, waves are reflected back into the computational domain. Therefore, it must be made sure that the boundary is sufficiently far away from regions where something happens, even for the computation of steady states. This makes the use of these conditions limited for unsteady flows.

To circumvent this problem, *nonreflecting boundary conditions* can be used [195] (also called *absorbing boundary conditions*). Essentially, we want outgoing waves to leave the computational domain without any disturbances reflecting backwards. This can be achieved by setting all incoming waves to zero. First, we use a formulation of the Euler equations using tangential (τ) and normal ($\mathbf{n}$) derivatives:

$$\partial_t \mathbf{u} + \partial_{\mathbf{n}} \mathbf{f}(\mathbf{u}) + \partial_\tau \mathbf{f}(\mathbf{u}) = \mathbf{0}.$$

We write the normal derivative in quasilinear form to obtain:

$$\partial_t \mathbf{u} + \mathbf{A_n} \partial_{\mathbf{n}} \mathbf{u} + \partial_\tau \mathbf{f}(\mathbf{u}) = \mathbf{0}.$$

Then we replace the Jacobian $\mathbf{A_n}$ by the matrix $\mathbf{A}^+_{\mathbf{n}} = \mathbf{R}\Lambda^+\mathbf{R}^{-1}$, which is obtained by diagonalizing $\mathbf{A_n}$, setting all negative eigenvalues to zero and transforming back. The matrix $\mathbf{A}^+_{\mathbf{n}}$ is also known from the theoretical analysis of the van Leer flux vector splitting. Thus the velocity of incoming waves is zero:

$$\partial_t \mathbf{u} + \mathbf{A}^+_{\mathbf{n}} \partial_{\mathbf{n}} \mathbf{u} + \partial_\tau \mathbf{f}(\mathbf{u}) = \mathbf{0}.$$

At an inlet edge between a ghost cell and an inner cell this equation is then discretized using first order upwinding. Note that the tangential component can be neglected in most applications. However, for example for buoyancy driven flows, a correct implementation of the tangential part is mandatory.

Another possibility are *characteristic boundary conditions*, where depending on whether we are at an inlet or outlet boundary, we prescribe three, respectively one value in the ghost cell neighbouring the boundary and use for the remaining variables the values from the computational domain. For the Navier-Stokes equations, we have to prescribe all values at an inflow boundary, but one less at an outflow boundary, mentioned in section 2.6. This way of setting the boundary conditions has been refined over the years by Nordström and others using the concept of summation by parts operators and the simultaneous approximation term (SAT), see [191] and the references therein.

However, even for the Navier-Stokes equations, often the conditions for the Euler equations are employed. Although it cannot be proven that this leads to a well-posed problem, it works quite well. The intuitive explanation is that away from turbulent structures and boundary layers, the second order terms can be neglected and thus, the Euler equations and their boundary conditions provide a very good approximation. We will now explain these in more detail.

As mentioned in section 2.6, the number of conditions to pose depends on the Mach number, respectively on the sign of the eigenvalues (2.30). Thus, for supersonic inflow, the farfield value determines the flux on the boundary, whereas for supersonic outflow, the value in the computational domain defines the boundary flux. For subsonic flow, conditions are derived using the theory of characteristics, respectively of Riemann invariants [221]. Given a value $\mathbf{u}_i$ in the cell next to the boundary, the value $\mathbf{u}_j$ in the ghost cell on the other side of the boundary with normal vector $\mathbf{n}$ has to be defined. The conditions obtained in this way are not unique, but depend on the components of the farfield values $\mathbf{u}_0$ used. At an inflow boundary, we obtain for the case that ρ_0 and $\mathbf{v}_0$ are prescribed

$$\begin{aligned}
\rho_j &= \rho_0, \\
\mathbf{v}_j &= \mathbf{v}_0, \\
c_j &= \frac{\gamma-1}{2}(v_{i_n} + 2c_i/(\gamma-1) - v_{0_n}).
\end{aligned}$$

The values for $\mathbf{m}_j$ and ρE_j are obtained from the values given above. For subsonic outflow in the case that the pressure p_0 is used, we have

$$\begin{aligned}
\rho_j &= \left(p_0 e^{-S_i/c_v}\right)^{1/(\gamma-1)} \\
\mathbf{v}_j &= \mathbf{v}_i + 2c_i/(\gamma-1)\mathbf{n} \\
c_j &= \sqrt{\gamma p_0/\rho_j}.
\end{aligned}$$

3.5.3 Periodic boundaries

Another possibility are periodic boundary conditions (2.27). These are an easy way to obtain long time numerical test problems that do not require a large computational domain. When using ghost cells, periodic boundary conditions are implemented by copying the value from the cells on the one periodic side to the ghost cells of the other periodic side and vice versa. With fluxes, these are computed based on the values in the cells on both sides of the periodic boundary. When using a cell-vertex method with a dual grid, things become slightly more complicated. One way is to define cells based on the vertices on the boundary, that then have volume on both sides. In this way, no fluxes along the periodic boundary need to be computed.

3.6 Source Terms

Regarding source terms, there are two basic approaches. The first one is to incorporate these into the computation of the fluxes. The other will be discussed in the next chapter and splits the source terms from the other terms to separately integrate these in time. One example are the well-balanced schemes of Greenberg and Leroux [75] or the Z-wave method of Bale et. al. [5].

3.7 Finite volume methods of higher order

The method as given so far uses a piecewise constant approximation to the solution $\mathbf{u}(\mathbf{x}, t)$ and therefore results in a method that can be at most of first order. This is not sufficient for practical applications, because the spatial resolution needed renders the schemes inefficient. Therefore, more accurate methods are necessary that need a smaller amount of degrees of freedom at as little additional cost as possible. A large number of approaches to obtain higher order have been suggested. The standard approach is the MUSCL scheme, that uses a reconstruction technique to obtain a linear representation $\mathbf{u}_i(t)$ of $\mathbf{u}(\mathbf{x}, t)$ in each cell. Others are WENO- and ADER schemes, as well as discontinuous Galerkin methods, which will be described in section 3.8. First, we will explain important results on higher order discretizations for conservation laws.

3.7.1 Convergence theory for higher order finite volume schemes

It turns out that a monotone scheme can be at most of first order [83]. Therefore, this requirement is too strong and needs to be relaxed. This has lead to the use of total variation dimininishing (TVD) schemes, which means hat the total variation of the numerical solution is nonincreasing with time, a property that is shared by the exact solutions in characteristic variables of hyperbolic equations. One point is that the TVD property implies that no new maxima and minima can be created, leading to nonoscillatory schemes.

It is possible to prove convergence results for a fixed time step Δt using the so called TV-stability property, which means that the total variation of the numerical solution is bounded, independently of the grid. If a scheme is consistent, conservative and TV-stable, then the solution converges in the L_1-norm to a weak solution of the conservation law, which is not necessarily the entropy weak solution. Note that to prove whether a scheme has the TVD property, it is not sufficient to consider the space discretization alone, but the time discretization has to be considered as well. For the sake of simplicity, we will only consider explicit Euler time integration for the remainder of this chapter and examine different time integration methods in the next chapter.

Unfortunately, it turns out that even with this relaxed property, TVD schemes are not the final answer. First, Godunov proved that a linear TVD scheme can be at most of first order [70]. For this reason, higher order schemes have to be nonlinear. Furthermore, they have to reduce to first order at spatial local maxima, as was proven by Osher and Chakravarty [152]. In multiple dimensions, as was proven by Goodman and LeVeque [71], TVD schemes are at most first order. Finally, on unstructured grids, not even a plane wave can be transported without increasing the total variation. Nevertheless, nonlinear TVD finite volume schemes using higher order ideas have become the workhorse in academia and industry in the form of the MUSCL schemes using reconstruction and limiters.

An alternative development are positive schemes, as made popular by Spekreijse [185], respectively local extrema diminishing schemes (LED) as suggested by Jameson [99]. This were developments in parallel to the development of TVD schemes that looked particularly interesting in the context of unsteady flows. However, it turns out that these schemes have the same restrictions as TVD schemes. Nevertheless, this led to the development of interesting flux functions like CUSP and SLIP, which are widely used in aerodynamic codes.

3.7.2 Reconstruction

The reconstruction procedure is based on the primitive variables $\mathbf{q} = (\rho, v_1, v_2, p)^T$, as this is numerically more stable than using the conservative variables [140]. At a given time t, the linear representation of a primitive variable $q \in \{\rho, v_1, ..., v_d, p\}$ in cell i with barycenter $\mathbf{x}_i$ is given by

$$q(\mathbf{x}) = q_i + \nabla q \cdot (\mathbf{x} - \mathbf{x}_i), \tag{3.18}$$

where q_i is the primitive value corresponding to the conservative mean values in Ω_i. The unknown components of ∇q represent the slopes and are obtained by solving a least square problem []. In the case of the Navier-Stokes equations, these slopes can sometimes be reused for the computation of the viscous fluxes.

For a cell-centered method on cartesian grids, the computation of the gradients is straightforward. If unstructured grids are employed, the following procedure is suitable. Let C be the closed polygonal curve that connects the barycenters of the neighbouring cells, see figure 3.6a. We then define the piecewise linear function q_c on C by setting

$$q_c(\mathbf{x}_j) = q_j$$

for all barycenters $\mathbf{x}_j$ of neighbouring cells. The least squares problem, which has to be solved for all primitive variables, is then:

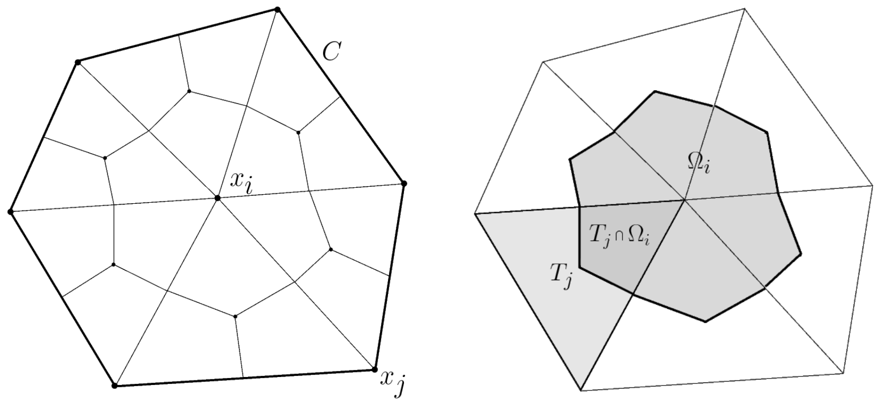


Figure 3.6: Left: The curve C used in the construction of the least squares problem for an unstructured cell. Right: The intersection of the primary triangle with a dual cell.

$$\min_{\nabla q \in \mathbb{R}^d} L(\nabla q) := \int_C (q(\mathbf{x}) - q_c(\mathbf{x}))^2 ds. \tag{3.19}$$

For a cell-vertex scheme, first the unique linear interpolants of the cell averages located at the nodes are computed in each primary cell T_j [142], as described in the section on viscous flows. Let these have the gradient ∇q_j. The gradient on the dual box Ω_i is then defined as

$$\nabla q = \frac{1}{|\Omega_i|} \sum_j \nabla q_j |T_j \cap \Omega_i|. \tag{3.20}$$

Again, this procedure has to be done for each primitive variable. Note that $|T_j \cap \Omega_i|$ is zero except for the few primary cells that have a part in the creation of the box Ω_i, see figure 3.6b. Thus, the reconstruction procedure is significantly easier to implement and apply for cell-vertex methods than for cell-centered methods in the case of unstructured grids.

3.7.3 Modification at the Boundaries

At a fixed wall, the boundary condition requires the flux to satisfy $\mathbf{v} \cdot \mathbf{n} = 0$ in the evaluation point, thus the slopes have to satisfy the same condition. For the cell-vertex method, nothing needs to be done, since there the values at the boundary will be zero in the first place and thus, the linear interpolants computed on the primary cells have the required property.

However, for the cell-centered method, we have to modify the original velocity slopes $(v_1)_{x_1}, (v_1)_{x_2}, (v_2)_{x_1}$ and $(v_2)_{x_2}$. We first define the unit vector $\eta = (\eta_1, \eta_2)^T$ to be the one pointing from the cell barycenter to the evaluation point on the wall and the unit vector $\vartheta = (\vartheta_1, \vartheta_2)^T = (-\eta_2, \eta_1)^T$ to be perpendicular to η. To ease the following construction, we now restrict the set of possible slope modifications $(\bar{v_1})_{x_1}, (\bar{v_1})_{x_2}, (\bar{v_2})_{x_1}, (\bar{v_2})_{x_2}$ satisfying $\bar{\mathbf{v}} \cdot \mathbf{n} = 0$ to those with a constant ϑ derivative. Thus we only modify the derivatives $(v_1)_\eta$ and $(v_2)_\eta$. This does not define the new slopes uniquely, therefore we require the new slopes to deviate from the old slopes minimally in the euclidian norm. We obtain the least squares problem:

$$\min(((\bar{v_1})_\eta - (v_1)_\eta)^2 + ((\bar{v_2})_\eta - (v_2)_\eta)^2 | (\bar{v_1})_{x_1}, (\bar{v_1})_{x_2}, (\bar{v_2})_{x_1}, (\bar{v_2})_{x_2} : \bar{\mathbf{v}} \cdot \mathbf{n} = 0).$$

With Δ being the euclidian distance from the cell barycenter to the evaluation point on the edge, we can write the modified velocity on the evaluation point as $\bar{\mathbf{v}} = \mathbf{v}_i + \Delta((v_1)_\eta, (v_2)_\eta)^T$, where $\mathbf{v}_i$ is the mean velocity vector of the cell and we obtain:

$$\bar{\mathbf{v}} \cdot \mathbf{n} = (\mathbf{v}_i + \Delta((v_1)_\eta, (v_2)_\eta)^T) \cdot \mathbf{n} = 0$$

$$\Leftrightarrow (\bar{v}_1)_\eta n_1 + (\bar{v}_2)_\eta n_2 = -\mathbf{v}_i \cdot \mathbf{n}/\Delta.$$

Inserting this condition directly into the least squares functional allows us to solve the minimization problem to obtain the solution $(\bar{v}_1)^\star_\eta, (\bar{v}_2)^\star_\eta$. Then we can compute the modified slopes via

$$(\bar{v}_1)_{x_1} = \eta_1(\bar{v}_1)^\star_\eta - \eta_2(v_1)_\vartheta, \quad (\bar{v}_2)_{x_1} = \eta_1(\bar{v}_2)^\star_\eta - \eta_2(v_2)_\vartheta,$$

$$(\bar{v}_1)_{x_2} = \eta_2(\bar{v}_1)^\star_\eta + \eta_1(v_1)_\vartheta \text{ and } (\bar{v}_2)_{x_2} = \eta_2(\bar{v}_2)^\star_\eta + \eta_1(v_2)_\vartheta.$$

3.7.4 Limiters

As mentioned before, to obtain a TVD scheme, the scheme can be at most of first order at shocks and local extrema. Thus, on top of the reconstruction scheme, a so called limiter function is needed. Typically a slope limiter ϕ is employed for the switching between first and higher order spatial discretization:

$$q(\mathbf{x}) = q_i + \phi \nabla q \cdot (\mathbf{x} - \mathbf{x}_i). \tag{3.21}$$

If the limiter function is zero, the discretization is reduced to first order, while it can be of second order for other values. In one dimension, conditions can be found for a limiter to result in a TVD scheme. For these, the limiter is taken as a function of the left and right slope and should be between zero and two, with the more precise shape given in figure 3.7a). Furthermore, the conditions for the resulting scheme to be second order are demonstrated in figure 3.7b). A large number of limiters has been suggested that fulfil these conditions, for example the superbee limiter or the van Albada limiter. On structured grids, these can be applied directionwise and it is possible to prove the TVD property.

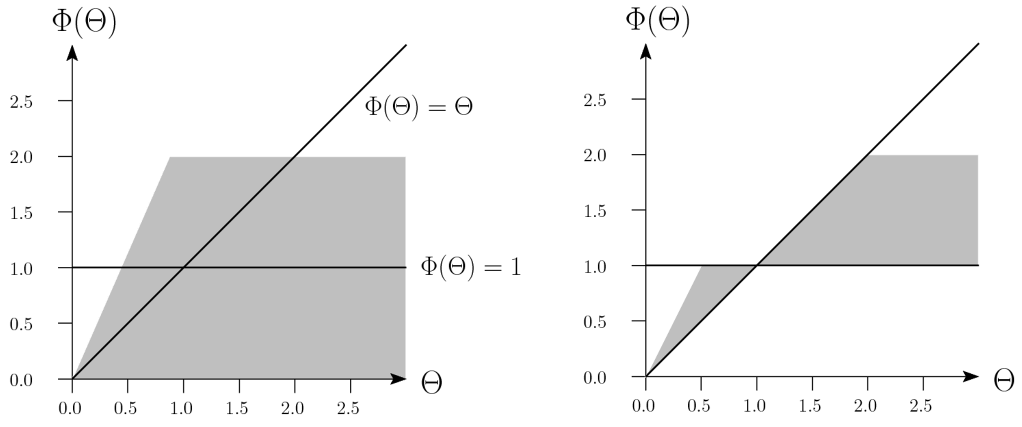


Figure 3.7: Left: Admissible region for a TVD slope limiter. Right: Admissible region for a 2nd order TVD slope limiter.

If the grid is unstructured, heuristic approaches must be used. We suggest either the Barth-Jesperson limiter [8] or the limiter proposed by Venkatakrishnan [212]. The latter is defined in cell i as follows:

$$\phi = \min_{j\in N(i)} \frac{(\Delta_{ij}^2+\epsilon)+2\Delta_{ij}\overline{\Delta}_{ij}}{\Delta_{ij}^2+2\overline{\Delta}_{ij}^2+\Delta_{ij}\overline{\Delta}_{ij}+\epsilon}. \tag{3.22}$$

Here, $\epsilon = 10^{-6}$ and

$$\Delta_{ij} = \begin{cases} 0, & \text{if } \overline{\Delta}_{ij}\Delta_{ij} < 0, \\ q_j - q_i, & \text{else,} \end{cases}$$

where the indicator $\overline{\Delta}_{ij}$ is given by

$$\overline{\Delta}_{ij} := q_{x_1}(x_{1_{ij}} - x_{1_i}) - q_{x_2}(x_{2_{ij}} - x_{2_i})$$

with the edge center $(x_{1_{ij}}, x_{2_{ij}})$.

Regarding boundaries, in the ghost cells at the inlet boundaries, the slopes and limiters are computed in the usual way. As the ghost cells are the definite border of the computational domain, no values beyond the ghost edges are interpolated or incorporated in any way, as is done for the fixed wall.

3.8 Discontinuous Galerkin methods

For example when trying to track vortices over a large time period, a scheme with high accuracy and little diffusion is necessary if the number of grid points is to be kept reasonable. A problem of finite volume methods, in particular in 3D, is the current inability to obtain methods that are efficient, of order higher than two and not extremely difficult to implement. A class of schemes that has received increasing research interest over the last twenty years are Discontinuous Galerkin (DG) schemes [114, 106, 90], because there is reason to believe that this class can replace finite volume schemes for industrial applications where highly accurate computations are necessary.

DG schemes can be seen as more natural extensions of first order finite volume schemes to higher orders than the reconstruction or WENO techniques mentioned earlier. Again, the solution is represented by a multivariate polynomial in each cell, leading to a cellwise continuous numerical solution. The specific polynomial is then determined using a Galerkin approach. There is a huge variety of methods based on this approach and a standard has only been established in parts. Furthermore, there are still a number of serious issues with DG methods that need to be solved before they are feasible for industrial applications [216]. In particular there is the treatment of curved boundaries, the efficient solution of the appearing systems inside an implicit time integration, as well as the formulation of appropriate limiter functions.

Starting point of a DG method is the weak form of the conservation law, whereby as test functions, polynomials ϕ from some test space are used:

$$\int_\Omega \mathbf{u}_t \phi d\Omega + \int_\Omega \nabla \cdot \mathbf{f}(\mathbf{u}, \nabla \mathbf{u}) \phi d\mathbf{s} = \mathbf{0}.$$

We then use integration by parts to obtain

$$\int_\Omega \mathbf{u}_t \phi d\Omega + \int_{\partial\Omega} \mathbf{f} \cdot \mathbf{n} \phi d\Omega - \int_\Omega \mathbf{f} \cdot \nabla \phi d\Omega = \mathbf{0}. \tag{3.23}$$

At this point, the solution is approximated in every cell Ω_i by a polynomial

$$\mathbf{u}_i^P(t; \mathbf{x}) = \sum_j \mathbf{u}_{i,j}(t) \phi_j(\mathbf{x}), \tag{3.24}$$

where $\mathbf{u}_{i,j}(t) \in \mathbb{R}^{d+2}$ are coefficients and ϕ_j are polynomial basis functions in $\mathbb{R}^d$ of up to degree p. Thus, by contrast to a finite volume method where we have just the $d + 2$ mean values $\mathbf{u}_i(t)$ as unknowns in cell Ω_i, there are now $d+2$ times the number of basis functions unknowns per cell for a DG method. We denote the dimension of this space by N.

Typically, the test functions are chosen to be from the same space. A specific DG method is obtained when we choose a precise polynomial basis, as well as the quadrature rules for the integrals in the scalar products, in particular the nodes and weights. Generally, there are two different types of basis polynomials, namely modal and nodal bases. A nodal basis is defined by a number of nodes, through which then Lagrange polynomials are defined. On the other hand, a modal basis is defined by functions only, a prime example would be monomials. Typically, a modal basis is hierarchical and some authors use the term hierarchical instead.

Obviously, the polynomial basis depends on the shape of the cell. Therefore, it is common for DG methods to restrict the possible shapes to for example quadrangles, triangles, tetrahedrons or cubes, then define the basis a priori for corresponding reference elements, e.g. unit quadrangles, triangles, tetrahedrons or cubes, to precompute as many terms as possible for that element. The basis for a specific cell is then obtained by a transformation from the reference cell. We will now demonstrate this for a curved quadrangle, the technique is similar for triangles.

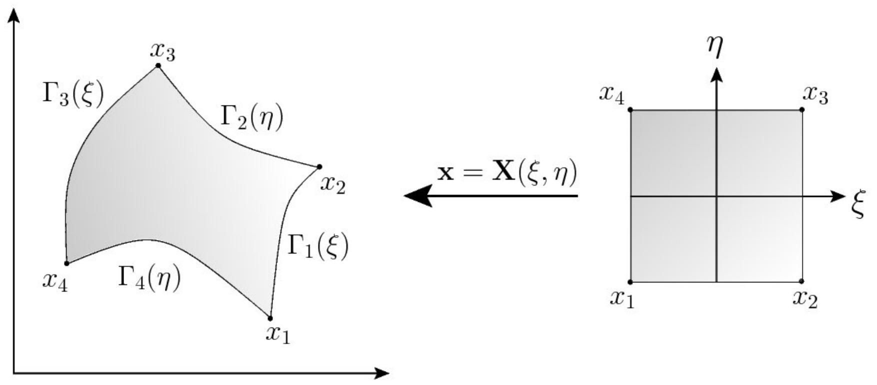


Figure 3.8: Illustration of the isoparametric mapping between an arbitrary quadrilateral cell and the reference element.

First, the cell Ω_i is transformed to a unit cell, for example $[0,1]^2$ with coordinates $\xi = (\xi_1, \xi_2) = (\xi, \eta)$ using an isoparametric transformation $\mathbf{r}$, which can be different for each cell. To this end, each of the four boundaries is represented by a polynomial $\Gamma_m(s)$, $m = 1, ..., 4$, $s \in [0,1]$ and the four corners are denoted by $\mathbf{x}_j$, $j = 1, ..., 4$. The isoparametric transformation can then be understood as mapping the four corners and curves onto their representative in the reference space and filling the domain in between by convex combinations of the curves opposite of each other:

$$\begin{aligned} \mathbf{r}_i(\xi,\eta) =& (1-\xi)\Gamma_1(\xi) + \eta\Gamma_3(\xi) + (1-\xi)\Gamma_4(\eta) + \xi\Gamma_2(\eta) \\ & - \mathbf{x}_1(1-\xi)(1-\eta) - \mathbf{x}_2\xi(1-\eta) - \mathbf{x}_3\xi\eta - \mathbf{x}_4(1-\xi)\eta. \end{aligned} \tag{3.25}$$

We thus obtain the new local equations

$$\mathbf{u}_t + \nabla_\xi \cdot \mathbf{f} = \mathbf{0}$$

with $\mathbf{u} = J\tilde{\mathbf{u}}$, $\tilde{u}$ being the values in the original coordinates and $\mathbf{f}_k = (\mathbf{v}_l \times \mathbf{v}_m) \cdot \tilde{\mathbf{f}}$ (k, l, m to be cyclic). Here, $\mathbf{v}_i = \partial\mathbf{r}/\partial\xi_i$, but formally embedded into $\mathbb{R}^3$ for $i = 1, 2$ and $\mathbf{v}_3 = (0,0,1)^T$. Furthermore, $J = \mathbf{v}_k \cdot (\mathbf{v}_l \times \mathbf{v}_m)$ is the factor coming from the transformation of the volume integrals, both of which depend on space if the mapping $\mathbf{r}$ is nonlinear. A derivation of this can be found in [114, Chapter 6]. Note that most of the above mentioned geometric quantities can be precomputed and stored directly after grid generation. For different reference cells, similar transformations can be employed.

On the reference cell Ω, we now define the quadrature rule to approximate the integrals in (3.23). Here, we choose Gaussian quadrature, such that polynomials of degree up to $2p$ are integrated exactly, which is well defined on the reference shapes with known nodes $\mathbf{x}_l$ and weights w_l:

$$\int_\Omega f(\mathbf{x})d\Omega \approx \sum_{j=1}^N f(\mathbf{x}_j)w_j. \tag{3.26}$$

Applying these quadrature rules in (3.23), we obtain for the different terms:

$$\int_\Omega \mathbf{u}_t^P \phi_k d\Omega = \int_\Omega \frac{d}{dt}\sum_{j=1}^N \mathbf{u}_j(t)\phi_j(\mathbf{x})\phi_k(\mathbf{x})d\Omega, \quad k = 1, ..., N \tag{3.27}$$

which, due to the quadrature rule being exact for polynomials up to degree $2p$, can be written as $\mathbf{M}\bar{\mathbf{u}}_t$. Here, the mass matrix $\mathbf{M} \in \mathbb{R}^{(d+2)N\times(d+2)N}$ is a block diagonal matrix with blocks in $\mathbb{R}^{(d+2)\times(d+2)}$ where the value on the diagonal of the block (j, k) is costant and given by

$$M_{jk} = \int_\Omega \phi_j\phi_k d\Omega, \quad j, k = 1, ..., N.$$

Furthermore, the vector of coefficients is given by

$$\bar{\mathbf{u}} = (\mathbf{u}_1^T(t), ..., \mathbf{u}_N^T(t))^T \in \mathbb{R}^{(d+2)N}.$$

For the volume integral, we obtain using (3.26):

$$\begin{aligned}\int_\Omega \mathbf{f}(\mathbf{u}^P)\cdot\nabla\phi_k d\Omega &= \int_\Omega \mathbf{f}\left(\sum_{j=1}^N \mathbf{u}_j(t)\phi_j(\mathbf{x})\right)\cdot\nabla\phi_k(\mathbf{x})d\Omega \quad k = 1, ..., N\\ &\approx \sum_{l=1}^N \mathbf{f}\left(\sum_{j=1}^N \mathbf{u}_j(t)\phi_j(\mathbf{x}_l)\right)\cdot\nabla\phi_k(\mathbf{x}_l)w_l \quad k = 1, ..., N.\end{aligned} \tag{3.28}$$

This can again be written in compact form as $\sum_{j=1}^d \mathbf{S}_j\bar{\mathbf{f}}_j$ with $\bar{\mathbf{f}}_j$ being the vector of evaluations of the function $\mathbf{f}_j$ at the quadrature nodes and $\mathbf{S}_j \in \mathbb{R}^{(d+2)N\times(d+2)N}$ being again a block diagonal matrix where on the diagonal of the block (k, l) we have the constant value

$$S_{kl} = \nabla\phi_k(\mathbf{x}_l)w_l, \quad k, l = 1, ..., N.$$

Finally, the boundary integral can be written as

$$\int_{\partial\Omega} \mathbf{f}\cdot\mathbf{n}\phi_k dS \approx \sum_{Faces}\sum_{l=1}^{\hat{N}} \mathbf{f}\left(\sum_{j=1}^N \mathbf{u}_j(t)\phi_j(\mathbf{x}_l)\right)\cdot\mathbf{n}\phi_k(\mathbf{x}_l)w_l, \quad k = 1, ..., N, \tag{3.29}$$

with $\hat{N}$ being the number of quadrature points on the boundary. Here, the compact form is $\mathbf{M}^S\bar{\mathbf{g}}_j$ with $\mathbf{M}^S \in \mathbb{R}^{(d+2)N\times(d+2)N}$ block diagonal with values

$$M^S_{kl} = \phi_k(\mathbf{x}_l)w_l$$

on the diagonal of the block (k, l). The vector $\bar{\mathbf{g}}_j$ consists of function evaluations at the quadrature nodes on the surface. As in the case of finite volume schemes, this has to be done using a numerical flux function, thus coupling neighboring cells together. Since the spatial resolution is obtained primarily through the high order polynomials, the specific flux function used is not as important as for finite volume schemes. For the inviscid fluxes, any of the previously discussed schemes can be used, whereas the choice of the viscous flux is slightly more complicated than before. This will be discussed in section 3.8.3.

Finally, we obtain the following ordinary differential equation for the coefficients in a reference cell:

$$\mathbf{M}\bar{\mathbf{u}}_t + \sum_{j=1}^{nFaces} \mathbf{M}^S_j\bar{\mathbf{g}}_j - \sum_{j=1}^{d} \mathbf{S}_j\bar{\mathbf{f}}_j = \mathbf{0}. \tag{3.30}$$

Note that the matrices $\mathbf{M}$, $\mathbf{M}^{\mathbf{S}}$ and $\mathbf{S}_j$ depend only on the basis and the geometry, not on $\mathbf{u}$ and can thus be precomputed after grid generation.

Obviously, the choice of basis is very important for the question of efficiency of the resulting methods. Essentially it turns out that for an efficient implementation of DG schemes, a nodal basis is more useful. This is because due to the use of Lagrange polynomials, a nodal basis allows the computation of the vectors $\mathbf{f}$ and $\mathbf{g}$ without the need for evaluating the ansatz function 3.24 at those points. In a similar fashion, some other approaches to obtain higher order methods also lead to a scheme that is similar to the nodal DG method, in particular the flux reconstruction schemes of Huynh [96] and the energy stable subfamily of schemes found by Vincent et. al. [215]. Here, we consider two particular types of DG schemes, namely the DG spectral element method (DG-SEM) approach of Kopriva [115] and the modal-nodal method suggested by Gassner et. al [65].

3.8.1 Polymorphic modal-nodal scheme

The modal-nodal approach allows for arbitrary element types and starts with a modal basis of monomials, that are based on the barycenters of reference cells. These are then orthonormalized using a Gram-Schmidt procedure. Thus, the basis is orthonormal and hierarchical, meaning that the basis functions can be sorted by the polynomial order. The dimension of the polynomial space is given by $N = (p + d)!/(p!d!)$, which amounts to 280 degrees of freedom in 3D for 5th order polynomials. Generally, this basis can be used to define a DG scheme as is. In this case, the vectors $\bar{\mathbf{f}}$ and $\bar{\mathbf{g}}$ contain a huge number of evaluations of the polynomial approximations (3.24). Therefore, this is combined with a nodal quadrature that reduces this computational effort significantly.

Thus, in addition a nodal basis is defined for the reference elements chosen, where we use Legendre-Gauss-Lobatto (LGL) points on the edges. For the interior points, an LGL-type approach is suggested in [65]. Other choices are possible, but the LGL-type approach leads to a very small condition number of the basis. The modal coefficients $\hat{\mathbf{u}}$ and the nodal coefficients $\tilde{\mathbf{u}}$ can then be related using a Vandermonde matrix $\mathbf{V}$, which contains the evaluations of the modal basis polynomials at the nodes defining the nodal basis:

$$\tilde{\mathbf{u}} = \mathbf{V}\hat{\mathbf{u}}. \tag{3.31}$$

If the number of basis functions is not identical, as is the case for triangles and tetrahedrons, this matrix is rectangular. Then, the backtransformation $\mathbf{V}^{-1}$ is defined in a least squares sense, as the pseudoinverse. Note that in both cases, these operators depend on the geometry and choice of basis alone, and thus can be precomputed and stored. All in all, we obtain the following ordinary differential equation for a reference element:

$$\hat{\mathbf{u}}_t = -\mathbf{V}^{-1}\tilde{\mathbf{M}}^{-1}\left(\sum_{i=1}^{nFaces}\tilde{\mathbf{M}}_i^S\tilde{\mathbf{g}}_i - \sum_{k=1}^{d}\tilde{\mathbf{S}}_k\tilde{\mathbf{f}}_k\right), \tag{3.32}$$

where the $\sim$ denotes terms based on the nodal basis.

3.8.2 DG Spectral Element Method

The DG-SEM requires quadrilateral cells, respectively hexahedral cells in three dimensions. However, hanging nodes as well as curved boundaries are possible. The method uses a nodal basis defined by Gaussian quadrature nodes. Thus, the dimension of the polynomial space is $N = (p+1)^d$. For the rest of the derivation, we will now assume a two dimensional problem, which simplifies the formulas significantly. The extension to three dimensions is straightforward. On the unit rectangular then approximation is then written as

$$\mathbf{u}^P(t;\xi,\eta) = \sum_{\mu,\nu=0}^{p}\mathbf{u}_{\mu\nu}(t)\phi_{\mu\nu}(\xi,\eta), \tag{3.33}$$

where we additionally use the ansatz

$$\mathbf{f}^P(\xi,\eta) = \sum_{\mu,\nu=0}^{p}\mathbf{f}_{\mu\nu}\phi_{\mu\nu}(\xi,\eta) \tag{3.34}$$

with

$$\mathbf{f}_{ij} = \mathbf{f}(\mathbf{u}_{ij},\nabla\mathbf{u}_{ij}), \quad \phi_{ij}(\xi,\eta) = l_i(\xi)l_j(\eta)$$

and Lagrange polynomials

$$l_j(\xi) = \Pi_{i=0,i\neq j}^{p}\frac{(\xi-\xi_i)}{\xi_j-\xi_i}$$

based on the nodes ξ_j, η_j of a Gauss quadrature. Note that we thus have to compute and store the metric terms from above per Gauss point per cell. Regarding the viscous terms, we need to compute gradients of the solution as well. We have for the gradient of the j-th component:

$$\nabla u_j^P(t,\xi,\eta) = \begin{pmatrix} \sum_{\mu,\nu=0}^p u_{\mu\nu_j}(t) l'_\mu(\xi) l_\nu(\eta) \\ \sum_{\mu,\nu=0}^p u_{\mu\nu_j}(t) l_\mu(\xi) l'_\nu(\eta) \end{pmatrix}.$$

The scalar products in (3.23) are approximated using Gauss quadrature with the weights w_i, w_j:

$$\int_{-1}^1 \int_{-1}^1 v(\xi,\eta) d\xi d\eta \approx \sum_{i,j=0}^p v(\xi_i,\eta_j) w_i w_j.$$

Thus, the final approximation in one cell is given by

$$\int_\Omega \mathbf{u}_t^P \phi_{ij} d\Omega + \int_{\partial\Omega} \mathbf{f}^P \cdot \mathbf{n} \phi_{ij} d\mathbf{s} - \int_\Omega \mathbf{f}^P \cdot \nabla \phi_{ij} d\Omega = \mathbf{0}, \quad i,j = 0,...,p.$$

Due to the choice of using the Lagrange polynomials based on the Gauss-Legendre points, the terms simplify significantly. We obtain

$$\int_\Omega \mathbf{u}_t^P \phi_{ij} d\Omega = \frac{d\mathbf{u}_{ij}}{dt} w_i w_j$$

and

$$\int_\Omega \mathbf{f}^P \cdot \nabla \phi_{ij} d\Omega = \sum_\mu \mathbf{f}_{1_{\mu j}} l'_i(\xi_\mu) w_\mu w_j + \sum_\nu \mathbf{f}_{2_{i\nu}} l'_j(\eta_\nu) w_i w_\nu.$$

Regarding the flux, the boundary integral is the sum of the four integrals on the edges of the unit cell. We have exemplary for the lower edge:

$$\int_{-1}^1 \mathbf{f}^P(\xi,-1) \cdot \mathbf{n} \phi_{ij}(\xi,-1) d\xi = -l_j(-1) \sum_{\mu,\nu=0}^p \mathbf{f}_{2_{\mu\nu}} l_\nu(-1) \int_{-1}^1 l_\mu(\xi) l_i(\xi) d\xi.$$

Approximating this using a the quadrature formula on the line, we obtain

$$\begin{aligned} \int_{-1}^1 \mathbf{f}^P(\xi,-1) \cdot \mathbf{n} \phi_{ij}(\xi,-1) dx &\approx -l_j(-1) \sum_{\mu,\nu=0}^p \mathbf{f}_{2_{\mu\nu}} l_\nu(-1) \sum_{a=0}^p l_\mu(\xi_a) l_i(\xi_a) w_i \\ &= -\mathbf{f}_2^P(\xi_i,-1) l_j(-1) w_i. \end{aligned}$$

Dividing by $w_i w_j$, we finally obtain

$$\begin{aligned} \frac{d\mathbf{u}_{ij}}{dt} &+ \left[\mathbf{f}_1^P(1,\eta_j) \frac{l_i(1)}{w_i} - \mathbf{f}_1^P(-1,\eta_j) \frac{l_i(-1)}{w_i} - \sum_\mu \mathbf{f}_{1_{\mu j}} \frac{l'_i(\xi_\mu) w_\mu}{w_i} \right] \\ &+ \left[\mathbf{f}_2^P(\xi_i,1) \frac{l_j(1)}{w_j} - \mathbf{f}_2^P(\xi_i,-1) \frac{l_j(-1)}{w_j} - \sum_\mu \mathbf{f}_{2_{i\mu}} \frac{l'_j(\eta_\mu) w_\mu}{w_j} \right] = \mathbf{0} \end{aligned} \qquad (3.35)$$

or in three dimensions:

$$
\begin{aligned}
\frac{d\mathbf{u}_{ijk}}{dt} &+ \left[\mathbf{f}_1^P(1,\eta_j,\zeta_k)\frac{l_i(1)}{w_i} - \mathbf{f}_1^P(-1,\eta_j,\zeta_k)\frac{l_i(-1)}{w_i} - \sum_\mu \mathbf{f}_{1_{\mu jk}}\frac{l_i'(\xi_\mu)w_\mu}{w_i}\right] \\
&+ \left[\mathbf{f}_2^P(\xi_i,1,\zeta_k)\frac{l_j(1)}{w_j} - \mathbf{f}_2^P(\xi_i,-1,\zeta_k)\frac{l_j(-1)}{w_j} - \sum_\mu \mathbf{f}_{2_{i\mu k}}\frac{l_j'(\eta_\mu)w_\mu}{w_j}\right] \\
&+ \left[\mathbf{f}_3^P(\xi_i,\eta_j,1)\frac{l_k(1)}{w_k} - \mathbf{f}_3^P(\xi_i,\eta_j,-1)\frac{l_k(-1)}{w_k} - \sum_\mu \mathbf{f}_{3_{ij\mu}}\frac{l_k'(\zeta_\mu)w_\mu}{w_k}\right] = \mathbf{0},
\end{aligned}
\tag{3.36}
$$

The sums in (3.35) and (3.36) can be computed using information given in that cell. However, the boundary term need to be coupled with the neighbours and since the numerical solution is discontinuous at the boundary by construction, again numerical flux functions are needed. We thus replace the boundary terms via

$$\mathbf{f}_1^P(1,\eta_j,\zeta_k) \approx \mathbf{f}^N(\mathbf{u}^P(1,\eta_j,\zeta_k); \hat{\mathbf{u}}^P(-1,\eta_j,\zeta_k); \mathbf{n}),$$

where $\hat{\mathbf{u}}^P$ corresponds to the polynomial from the neighboring cell.

3.8.3 Discretization of the viscous fluxes

More difficulty is posed by the diffusive terms, since the averaging procedure used in the finite volume case is unstable in this context. Several options to circumvent this problem are available, for example the Bassi-Rebay-flux in version 1 and 2 [11], local DG of Cockburn and Shu [39] and the compact DG flux (CDG) of Persson and Perraire [156]. We will use the diffusive Generalized Riemann Problem (dGRP) flux of Gassner et. al. [63, 64], which, as CDG, has the advantage of using a small discretization stencil. To derive this flux, we first have to go back to the volume integral in (3.23) and rewrite the viscous part (2.17), using the notation $\mathbf{f}^v = \mathbf{D}(\mathbf{u})\nabla\mathbf{u}$ and a second integration by parts, as

$$\int_\Omega \mathbf{D}(\mathbf{u})\nabla\mathbf{u}\cdot\nabla\phi d\Omega = \int_{\partial\Omega} \mathbf{u}\cdot((\mathbf{D}(\mathbf{u})^T\nabla\phi)\cdot\mathbf{n}d\mathbf{s} - \int_\Omega \mathbf{u}\cdot(\nabla\cdot\mathbf{D}(\mathbf{u})^T\nabla\phi))d\Omega. \tag{3.37}$$

We now introduce the adjoint flux

$$\mathbf{f}^{vv}(\mathbf{u},\nabla\phi) := \mathbf{D}^T(\mathbf{u})\nabla\phi, \tag{3.38}$$

which is later used to make sure that the discretization is consistent with the adjoint problem. A failure to do so results in either nonoptimal convergence rates or higher condition numbers [2]. To obtain a more efficient evaluation of the second term in (3.37) we use a third integration by parts

$$\int_\Omega \mathbf{u}\cdot(\nabla\cdot\mathbf{D}(\mathbf{u})^T\nabla\phi))d\Omega = \int_{\partial\Omega}[\mathbf{u}\cdot(\mathbf{n}\cdot\mathbf{f}^{vv}(\mathbf{u},\nabla\phi))]_{INT}ds - (\mathbf{f}^v,\nabla\phi), \tag{3.39}$$

where the subscript INT refers to an interior evaluation of the respective function at the boundary. Combining (3.37) and (3.39) in (3.23) we obtain

$$(\mathbf{u}_t,\phi) + \int_{\partial\Omega}\mathbf{f}\cdot\mathbf{n}\phi ds - (\mathbf{f},\nabla\phi) + \int_{\partial\Omega}\mathbf{h}(\mathbf{u},\mathbf{n},\nabla\phi)ds = \mathbf{0} \tag{3.40}$$

with the additional diffusion flux

$$\mathbf{h}(\mathbf{u},\mathbf{n},\nabla\phi) := \mathbf{u}\cdot(\mathbf{n}\cdot\mathbf{f}^{vv}(\mathbf{u},\nabla\phi)) - [\mathbf{u}\cdot(\mathbf{n}\cdot\mathbf{f}^{vv}(\mathbf{u},\nabla\phi))]_{INT}. \tag{3.41}$$

We can now formulate the dGRP flux by defining the numerical fluxes used in the computation of the boundary integrals in (3.40). For the viscous fluxes, this is derived from the viscous Riemann problem. If we assume that the flux has been rotated into normal direction (which is possible since the adjoint flux is rotationally invariant as well) and using the characteristic length scale

$$\Delta x_{ij} = \frac{\min(|\Omega_i|,|\Omega_j|)}{|\partial\Omega_{ij}|},$$

a constant η and the averaging operator

$$\{\mathbf{u}\} = 0.5(\mathbf{u}_L + \mathbf{u}_R), \tag{3.42}$$

the dGRP flux for the viscous term is given by

$$\mathbf{f}^{dGRP}(\mathbf{u}_L,\mathbf{u}_R) = \mathbf{f}_1^v\left(\{\mathbf{u}\},\left(\frac{\eta}{\Delta x_{ij}}(\mathbf{u}_L-\mathbf{u}_R)+\{\mathbf{u}_n\},\{\mathbf{u}_t\}\right)^T\right), \tag{3.43}$$

where $\mathbf{u}_t$ is the tangential component of $\mathbf{u}$. The constant η can be considered as a penalization of the jumps and is given by

p	1	2	3	4	5
β^*	1.5	0.7	0.35	0.25	0.13

Table 3.1: Parameter β^* in the dGRP scheme.

$$\eta = \frac{2p+1}{\sqrt{\pi}\beta^*(p)}$$

with β^* a real number which becomes smaller with increasing order p, see table 3.1. Furthermore, for the adjoint flux (3.41), the approximation

$$\mathbf{h} = \frac{1}{2}(\mathbf{u}_L-\mathbf{u}_R)\cdot(\mathbf{f}^{vv}(\{\mathbf{u}\},\nabla\Phi_L)\cdot\mathbf{n}) \tag{3.44}$$

is used.

3.9 Convergence theory for DG methods

In the case of DG methods, the theory does not use the total variation stability to prove convergence, as is the case for finite volume methods. Instead, energy stability is employed. Thus, for a wide range of equations, convergence of higher order can be proved, see e.g. [90]. In particular, for scalar equations on cartesian and particular structured triangular grids, the optimal convergence order of $p+1$ can be proved, whereas on unstructured grids, at most $p + 1/2$ is obtained.

Zhang and Shu could prove using the energy method that for smooth solutions of a symmetrisable hyperbolic system in multiple dimensions, a DG scheme on triangles using an appropriate (SSP, as explained later) time integration scheme and an appropriate flux, converges in L_2 with a spatial order of $p + 1/2$ [226].

As for finite volume schemes, these results are only valid under a time step restriction, which turns out to be more severe and to depend on the polynomial degree. This will be discussed in section 4.2.

3.10 Spatial Adaptation

Though important, the topic of spatial adaptation is not a focus of this book. We will nevertheless give a few pointers to the important concepts in this field. Thereby, we concentrate on methods that can be used on unstructured grids. These lead to a huge amount of freedom in grid generation, which can be used in the adaptation strategy. The major problem is therefore to find a good error estimator.

As mentioned before, the convergence theory for nonlinear systems of conservation laws for both finite volume and DG methods is lacking and therefore, rigorous error estimates similar to theorem 2 do not exist for the interesting methods and equations. In fact, it is known for the Euler equations that nonunique solutions may exist and for the Navier-Stokes equations, uniqueness has not been proven for general initial conditions. Therefore, by contrast to finite element methods for elliptic PDEs, the grid adaptation has to be based on more or less heuristical error estimators.

For steady state computations, a large amount of literature is available on grid adaptation. There, it is clear that any error in the solution is due to the space discretization. Therefore, grid adaptation can be done by starting on a coarse grid, computing the steady state on that grid, estimating the error of that solution and iterating the last two steps. For unsteady problems in the context of the method of lines, it is not straightforward to separate the spatial discretization error. Note that due to the sequential nature of the method of lines, the time discretization error for a given ordinary differential equation defined by the spatial discretization can be estimated quite well by the time integration method, as will be described in the next chapter. A remedy to this are space-time discretization methods that will be briefly discussed in section 4.9. The alternative way typically used is to perform a grid refinement every couple of time steps.

Error estimators for steady state problems typically use the assumption that the right hand side of the ODE vanishes, respectively that $\dot{\mathbf{u}} = \mathbf{0}$. First of all, the residual can be used, although it is known that this can be a misleading indicator for the local error for nonlinear problems. Then, there is the popular method of gradient based grid adaptation. There, the assumption is made that large gradients correspond to large local discretization errors, which is valid for smooth solutions, e.g. subsonic flows. For transonic or supersonic flows which typically contain shocks, gradients can even increase with grid adaptation, leading to a better accuracy, but a worse error estimation. Nevertheless, this leads to reasonably good results in practice.

Finally, there are goal oriented methods that adapt the grid not to decrease the error in the numerical approximation to the PDE, but to decrease errors in a user specified, typically nonlinear functional $J(\mathbf{u})$. The functional reflects a quantity that the user is particularly interested in, for example lift over drag. This is useful, since often the solution of the PDE is of minor interest and only the mean to compute the value of a functional. The most widely used technique is the dual weighted residual approach [12]. There, the error in the value of the functional $J(\mathbf{u}^*) - J(\mathbf{u}_h)$ is estimated, where $\mathbf{u}^*$ is the solution and $\mathbf{u}_h$ a discrete approximation. This is done by solving a linearized form of the so called dual or adjoint problem. This makes the grid adaptation process rather costly, but if successful, leads to a powerful refinement strategy [84]. Another question is if a functional other than the residual can be found that leads to an overall good solution to the complete problem. To this end, Fidkowski and Roe suggest to use an entropy functional [61].

Chapter 4

Time Integration Schemes

After the discretization in space, when combining the equations for one cell in one coupled system, we obtain a large system of ordinary differential equations (ODEs) or more precisely, initial value problems (IVP)

$$\frac{d}{dt}\underline{\mathbf{u}}(t) = \underline{\mathbf{f}}(t, \underline{\mathbf{u}}(t)), \quad \underline{\mathbf{u}}(t_0) = \underline{\mathbf{u}}^0, \quad t \in [t_0, t_{end}], \tag{4.1}$$

where $\underline{\mathbf{u}} \in \mathbb{R}^m$ is a vector containing all the unknowns from all grid cells. Very often in CFD, the function $\underline{\mathbf{f}} : \mathbb{R}^{m+1} \to \mathbb{R}^m$ defined by the spatial discretization has no explicit dependence on t and then we obtain an autonomous IVP:

$$\frac{d}{dt}\underline{\mathbf{u}}(t) = \underline{\mathbf{f}}(\underline{\mathbf{u}}(t)), \quad \underline{\mathbf{u}}(0) = \underline{\mathbf{u}}^0, \quad t \in [t_0, t_{end}]. \tag{4.2}$$

There is a huge variety of numerical methods to solve problems of this type, see for example the classic text books [81, 82]. Typically, these provide a number of approximations $\underline{\mathbf{u}}^n \approx \underline{\mathbf{u}}(t_n)$ at discrete times $t_0, t_1, ..., t_{end}$. Using the generating function Φ, they can be written as:

$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^n + \Delta t \Phi, \quad t_{n+1} = t_n + \Delta t.$$

If Φ is a function of data points at past times, it is called a multistep method, otherwise it is a onestep method. Multistep methods need some sort of starting procedure, for example a onestep method. Furthermore, a scheme is called implicit if it incorporates the unknown data $\underline{\mathbf{u}}^{n+1}$, otherwise it is called explicit. Implicit schemes require solving an equation system in every step. In this chapter, we will assume that these can be solved to whatever accuracy is needed. Methods for doing so, as well as the problems appearing there, are discussed in the next chapter. The prototypical schemes are the explicit Euler method

$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^n + \Delta t \mathbf{f}(\underline{\mathbf{u}}^n) \tag{4.3}$$

and the implicit Euler method

$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^{n} + \Delta t \mathbf{f}(\underline{\mathbf{u}}^{n+1}). \tag{4.4}$$

4.1 Order of convergence and order of consistency

There is a large number of different time integration methods. Two basic important properties necessary to describe and understand the behavior of a method are the order of convergence and the order of consistency.

Definition 1 *The local truncation error of a method is defined as the difference between the exact solution of an IVP and the numerical solution obtained after one step if exact initial data is used:*

$$l = u(t_{n+1}) - u(t_n) - \Delta t \Phi(u(t)). \tag{4.5}$$

Definition 2 *A method is called consistent of order p if for any right hand side $f \in C^{p+1}$, the norm of the local truncation error (4.5) is $\mathcal{O}(\Delta t^{p+1})$.*

If a method is consistent, this means that the local truncation error will converge to zero for Δt to zero or otherwise put that the method actually solves the correct problem. This property is about one time step, which means that we can actually measure it. However, a more important property is if it is possible that the error after a large number of steps still has something to do with the solution.

Definition 3 *A method is called convergent of order p if for any right hand side $f \in C^{p+1}$, the norm of the error $e_n := u(t_n) - u_n$ is $\mathcal{O}(\Delta t^{p})$.*

The order of convergence is an important property of any method and the idea is that higher order methods will allow for higher time steps for a given error tolerance, but a smaller cost per unit step.

It should be noted that due to the use of Landau symbols in the definition, consistency and convergence are defined only for $\Delta t \to 0$, whereas practical numerical calculations are always carried out for Δt well away from zero. Therefore consistency alone is not sufficient to obtain convergence and we need a notion of stability in addition to that.

4.2 Stability

Roughly speaking, a method is called stable, if it is robust with respect to the initial data. This implies that rounding errors do not accumulate or more precise that the error remains bounded for t to infinity for a fixed time step size. This is very difficult to establish for a general right hand side, which is why a large of number of stability properties exist for special equations.

4.2.1 The linear test equation, A- and L-stability

Important insights can be gained already in the scalar case using the Dahlquist test equation

$$\frac{d}{dt}u = \lambda u,\ u(0) = 1. \tag{4.6}$$

For a λ with negative real part, the exact solution decays to zero for t to infinity. Consequently a method is stable if the numerical solution to this problem remains bounded. Note that this depends on the step size Δt. If we consider only schemes with fixed step sizes, the set of all complex numbers $\Delta t\lambda$ for which the method is stable is called the stability region of the method. This stability region differs widely from method to method, see [82]. For the explicit Euler method, it is easy to show that the stability region is the circle around -1 with radius 1, whereas for the implicit Euler method it is the complete complex plane minus the circle around 1 with radius 1.

Since for a λ with positive real part, the solution is unbounded, it can not be expected that the error remains bounded. Therefore, the left half of the complex plane plays a special role in the evaluation of the stability of a numerical method. This gives rise to the notion of A-stability:

Definition 4 *A scheme is called A-stable, if the stability region contains the left complex half plane.*

Obviously, the implicit Euler method is A-stable, whereas the explicit Euler method is not. Generally speaking, explicit methods are not A-stable, whereas implicit methods can be.

This property is rather strong, but turns out to be insufficient for the solution of CFD problems, because A-stability is not able to prevent oscillations due to very high amplitude eigenmodes. Therefore, the following more strict property is required. Here, $R(z)$ is the stability function of a onestep method for the test equation, defined via

$$u^{n+1} = R(\Delta t\lambda)u^n. \tag{4.7}$$

In other words, the stability region of a onestep method is the set of all z with $|R(z)| \leq 1$.

Definition 5 *A scheme is called L-stable, if it is A-stable and furthermore, the stability function satisfies*

$$\lim_{z\to\infty} R(z) = 0.$$

These stability properties are derived using a scalar linear test equation and the question is what the relevance is for more complex equations. For a linear system with constant coefficients

$$\frac{d}{dt}\underline{\mathbf{u}}(t) = \mathbf{A}\underline{\mathbf{u}},$$

it can be proved that if an RK method is A-stable, then it is unconditionally contractive in the 2-norm, if the matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ is normal and the real part of its eigenvalues is negative. For a nonnormal matrix $\mathbf{A}$, this is not true and we must expect a more severe constraint on the time step.

For nonlinear equations, even less can be said. Therefore, more general stability properties aimed at nonlinear equations like AN-stability have been suggested. However, A- and L-stability seem to be sufficient for the cases discussed here.

4.2.2 TVD stability and SSP methods

The stability properties discussed so far are aimed at general equations and do not take into account special properties of the IVPs considered here, which arise from the space discretization of flow problems. As remarked in the last chapter (see section 3.7.1), there is an interesting form of stability that can be used to prove convergence of finite volume methods. This is the so called strong stability [73], originally suggested in [175] and [176] under the name of TVD-stability. Since then, a large number of articles have been published on this topic, see for example the recent review articles [72] and [110]. To define strong stability, we assume that there exists a value Δt_{EE}, such that the explicit Euler method is stable for a certain class of functions $\underline{\mathbf{f}}$ in the following sense for all $0 \leq \Delta t \leq \Delta t_{EE}$:

$$\|\underline{\mathbf{u}}^n + \Delta t\underline{\mathbf{f}}(\underline{\mathbf{u}}^n)\| \leq \|\underline{\mathbf{u}}^{n+1}\| \tag{4.8}$$

for an appropriate norm $\|\cdot\|$. This is in line with the requirement that the total variation does not increase with time, which corresponds to our definition of TVD schemes. The following definition carries this over to higher order time integration methods.

Definition 6 *An s-step method is called strong stability preserving (SSP) on a certain class of functions with strong stability constant $c > 0$, if it holds that*

$$\|\underline{\mathbf{u}}^{n+1}\| \leq \max\{\|u^n\|, \|u^{n-1}\|, ..., \|u^{n+1-s}\|\}$$

for any time step $\Delta t \leq c\Delta t_{EE}$.

This can be related to A-stability in that a method is A-stable, if it is unconditionally SSP for the linear test equation with the real part of $\lambda \in \mathbb{C}^-$ in the norm $|\cdot|$. The crucial question is now, if we can construct methods with large c. There is good news in that it turns out that the implicit Euler method is unconditionally SSP. However, for any general linear method of order greater than one, the SSP constant is finite. Furthermore, explicit RK methods or SDIRK methods that are SSP can have an order of at most 4. A more precise bound has not been proved so far, however, convincing numerical evidence obtained by an optimization software shows the following results [110]. For a second order SDIRK method with s stages, the bound is $2s$, for third order, it is $s - 1 + \sqrt{s^2 - 1}$. For higher orders, the bounds obtained are roughly speaking $2s$. For linear multistep methods of order greater than one, the SSP constant is smaller or equal to two.

These SSP coefficients are well below what is needed to make implicit methods competitive. Therefore, if using an implicit method, we do not know for sure that our numerical method is TVD. This does not seem to matter at all in practice. Nevertheless, for problems where explicit methods are faster than implicit ones, SSP methods should be preferred to be on the safe side.

4.2.3 The CFL condition, Von-Neumann stability analysis and related topics

The stability conditions so far have been derived using ordinary differential equations only. If we consider linear partial differential equations instead, more insight can be gained. The conditions obtained in this way are less strict than the SSP conditions, but nevertheless useful in practice. Furthermore, the maximal stable time step Δt_{EE} for the explicit Euler method used in the SSP theory can be determined.

First of all, for the hyperbolic inviscid terms, we know that the timestep size has to satisfy the Courant-Friedrichs-Levy (CFL) condition that the domain of dependence of the solution has to contain the domain of dependence of the numerical method [41]. This is illustrated in figure 4.1.

The CFL condition is automatically satisfied by implicit schemes, regardless of the time step. For finite volume methods for one dimensional equations with explicit time integration, we obtain the constraint

$$\Delta t < CFL_{max} \cdot \frac{\Delta x}{\max_{k,|\mathbf{n}|=1} |\lambda_k(\mathbf{u}, \mathbf{n})|}, \tag{4.9}$$

with the maximal CFL number CFL_{max} depending on the method, for example 1.0 for the explicit Euler method. The λ_k are the eigenvalues (2.30) of the Jacobian of the inviscid flux. When comparing this constraint to the one that would come out of applying the eigenvalue analysis as for the linear test equation, it turns out that this constraint is twice as strong, meaning that the time step is twice as small.

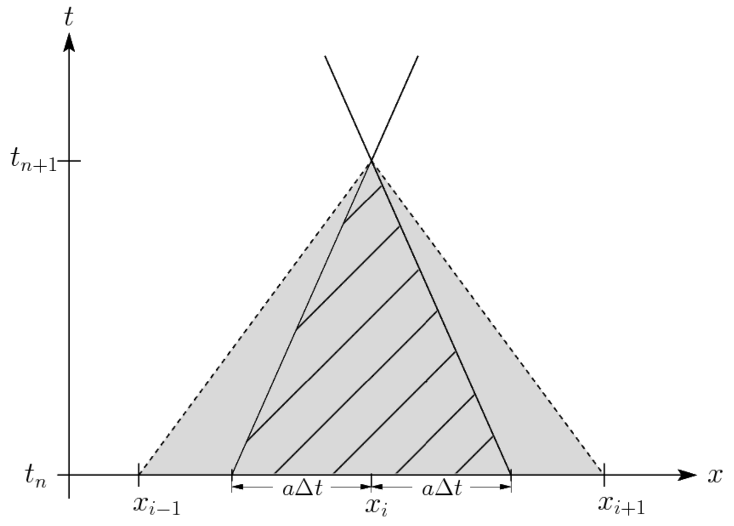


Figure 4.1: Illustration of the CFL condition for a linear equation: the exact solution in the point (x_i, t_{n+1}) can be influenced by the points in the shaded area. The numerical domain of dependence (the grey area) has to contain the shaded area for the scheme to be stable, resulting in a constraint on $\Delta t/\Delta x$.

It can be shown that the condition (4.9) is the result of a linear stability analysis for an upwind discretization, namely the von Neumann stability analysis. This is, besides the energy method, the standard tool for the stability analysis of discretized partial differential equations. To this end, the equation is considered in one dimension with periodic boundary conditions. Then, the discretization is applied and Fourier data is inserted into the approximation. The discretization is stable if no component is amplified, typically leading to conditions on Δt and Δx. Unfortunately, this technique becomes extremely complicated when looking at unstructured grids, making it less useful for these. For linear problems, the analysis results in necessary and sufficient conditions. This means that the CFL condition is sharp for linear problems.

For nonlinear equations the interactions between different modes are more than just superposition. Therefore, the von Neumann stability analysis leads to necessary conditions only. Typically, there are additional stability constraints on top of these. For the case of the viscous Burger's equation, this has been demonstrated in [131]. For general equations, no complete stability analysis is known.

If additional parabolic viscous terms are present, like in the Navier-Stokes equations, the stability constraint changes. For a pure diffusion equation, resp. the linear heat equation, this is sometimes called the DFL condition and for a finite volume method given by

$$\Delta t < \frac{\Delta x^2}{2\epsilon}, \tag{4.10}$$

where ϵ is the diffusion coefficient. Here, the dependence on the mesh width is quadratic and thus, this condition can be much more severe for fine grids than the CFL condition.

One way of obtaining a stable time integration method for the Navier-Stokes equations is to require the scheme to satisfy both the CFL and DFL conditions. However, this is too severe, as seen in practice and by a more detailed analysis. For example, the restriction on the time step in theorem 3 is of the form

$$\Delta t < \frac{\alpha^3 \Delta x^2}{\alpha \max_{k,|\mathbf{n}|=1} |\lambda_k(\mathbf{u},\mathbf{n})| \Delta x + \epsilon}. \tag{4.11}$$

Here, α is a grid dependent factor that is roughly speaking closer to one the more regular the grid is (see [150] for details). A similar bound has been found in praxis to be useful for the Navier-Stokes equations:

$$\Delta t < \frac{\Delta x^2}{\max_{k,|\mathbf{n}|=1} |\lambda_k(\mathbf{u},\mathbf{n})| \Delta x + 2\epsilon}.$$

In the case of a DG method, both time step constraints additionally depend on the order of the polynomial basis in that there is an additional factor $1/(2N+1)$ for the CFL condition and of $1/N^2$ for the DFL condition. The dependence of the stability constraint on the choice of the viscous flux has been considered in [105].

In more than one space dimension, the situation is, as expected, more difficult. It is not obvious which value to choose in a cell for Δx and no stability analysis is known. A relation to determine a locally stable time step in a cell Ω that works on both structured and unstructured grids for finite volume schemes is the following:

$$\Delta t = \sigma \frac{|\Omega|}{\lambda_{c,1} + \lambda_{c,2} + \lambda_{c,3} + 4(\lambda_{v,1} + \lambda_{v,2} + \lambda_{v,3})}.$$

Here, $\lambda_{c,i} = (|v_i| + c)|s_i|$ and

$$\lambda_{v,i} = \max\left(\frac{4}{3\rho}, \frac{\gamma}{p}\right) \frac{\mu}{Pr} \frac{|s_i|}{|\Omega|},$$

where s_i is a projection of the control volume on the planes orthogonal to the x_i direction [214]. The parameter σ corresponds to a CFL number and has to be determined in practice for each scheme, due to the absence of theory. Finally, a globally stable time step is obtained by taking the minimum of the local time steps.

4.3 Stiff problems

Obviously, implicit methods are more costly than explicit ones. The reason they are considered nevertheless are stiff problems, which can loosely be defined as problems where implicit methods are more efficient than explicit ones. This can happen since broadly speaking, implicit schemes have better stability properties than explicit ones. If the problem is such, that a stability constraint on the time step size leads to time steps much smaller than useful to resolve the physics of the system considered, the additional cost per time step can be justified making implicit schemes the method of choice. To illustrate this phenomenon, consider the system of two equations

$$\begin{aligned} x_t &= -x, \\ y_t &= -1000y. \end{aligned}$$

The fast scale quantity $y(t)$ will decay extremely fast to near zero and thus, large time steps that resolve the evolution of the slow scale quantity $x(t)$ should be possible. However, a typical explicit scheme will be hampered by stability constraints in that it needs to chose the time step according to the fastest scale, even though this has no influence on the solution after an initial transient phase. This example illustrates one possibility of characterising stiffness: we have a large Lipschitz constant of our function, but additionally eigenvalues of small magnitude. In this case, the longterm behavior of the solution is much better characterised by the largest real part of the eigenvalues, instead of the Lipschitz constant itself [47]. This is connected to the logarithmic norm. Another possibility of obtaining stiffness are stiff source terms. Note that if stiffness manifests itself also depends on the initial values chosen, as well as on the maximal time considered, which is why it is better to talk of stiff problems instead of stiff equations.

The Navier-Stokes equations have similar properties as the above system with both slow and fast scales present. First of all for the Euler equations, there are fast moving acoustic terms which correspond to the largest eigenvalue. Then, there are the significantly slower convective terms, which correspond to the convective eigenvalue $|v|$. The time step size of explicit schemes will be dictated by the acoustic eigenvalue, even if the physical processes of interest usually live on the convective scale. This property becomes extreme for low Mach numbers, when the convective eigenvalue approaches zero and the quotient between these two eigenvalues becomes very large, e.g. stiffness increases.

In the case of the Navier-Stokes equations with a bounding wall, we have additionally the boundary layer. This has to be resolved with extremely fine grid spacing in normal direction, leading to cells which are several orders of magnitude smaller than for the Euler equations. For an explicit scheme, the CFL condition therefore becomes several orders of magnitude more restrictive, independent of the flow physics. Furthermore, large aspect ratios are another source of stiffness, sometimes called geometric stiffness. Therefore, implicit methods play an important role in the numerical computation of unsteady viscous

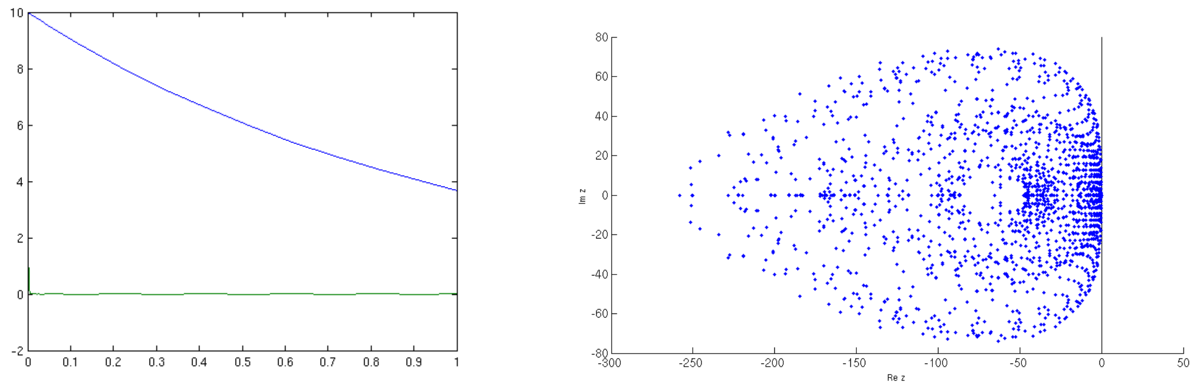


Figure 4.2: Solution of the stiff example equation with initial condition (10, 10) (left); Eigenvalues for a 2D Navier-Stokes DG discretization with $Re = 100$, 6×6 mesh, 4th order in space (right).

flows. If no boundary layer is present, this problem is less severe and it may be that explicit methods are still the methods of choice.

4.4 Backward Differentiation formulas

A class of time integration schemes that belongs to the multistep methods are backward differentiation formulas, short BDF. These approximate the solution by a p-th order polynomial that interpolates the values $\underline{\mathbf{u}}^{n+1-k}$ at the points t_{n+1-k}, $k = 0, ..., p$. Since both $\underline{\mathbf{u}}^{n+1}$ and $\underline{\mathbf{f}}(\underline{\mathbf{u}}^{n+1})$ are unknown, it is additionally required that the polynomial fulfills the differential equation at t_{n+1}. Thus, to determine the new approximation $\underline{\mathbf{u}}_{n+1}$, one nonlinear equation system has to be solved:

$$\sum_{i=1}^{p+1} \alpha_i \underline{\mathbf{u}}^{n+1-i} = \Delta t \underline{\mathbf{f}}(\underline{\mathbf{u}}^{n+1}). \tag{4.12}$$

Here, the coefficients α_i are defined via the interpolation polynomial and thus depend on the step size history.

The resulting methods are of order p, but unstable for $p > 6$. For $p = 1$, the implicit Euler method is obtained. The method BDF-2, obtained for $p = 2$, for a fixed time step size Δt is

$$\frac{1}{\Delta t}\left(\frac{3}{2}\underline{\mathbf{u}}^{n+1} - \frac{4}{2}\underline{\mathbf{u}}^{n} + \frac{1}{2}\underline{\mathbf{u}}^{n-1}\right) + \underline{\mathbf{f}}(\underline{\mathbf{u}}^{n+1}) = 0. \tag{4.13}$$

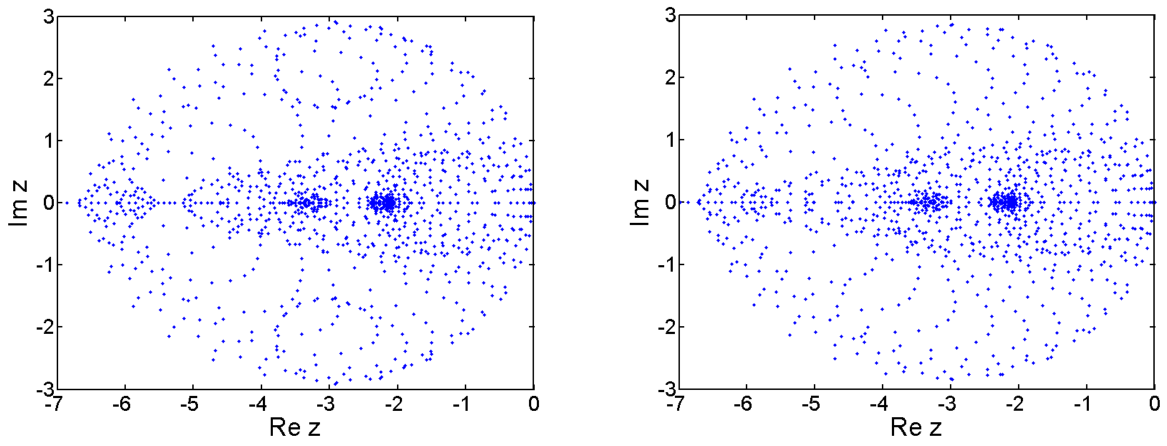


Figure 4.3: Eigenvalues for a 2D finite volume discretization for Euler (left), and Navier-Stokes with $Re = 1000$ (right).

This method is A-stable and also L-stable. For higher orders, this is impossible due to the second Dahlquist barrier. There, the methods are A(α)-stable, with decreasing α. As shown in figure 4.3 (right), this is problematic. As for the SSP property, linear multistep methods of order greater than one have an SSP constant of two or smaller [72].

In the case of varying time step sizes, the coefficients of the method vary as well. In practice, this is neglected and thus, an additional time integration error is introduced. BDF methods with varying time step sizes are used in the well know ODE solver DASSL of Petzold et. al. [32], as well as in SUNDIALs [92]. BDF methods, in particular BDF-2, are often used in the engineering community, with the reason given that you get a second order A-stable method that needs only one nonlinear system.

4.5 Runge-Kutta methods

By contrast to multistep methods, the by far most widely used class of onestep methods are the Runge-Kutta (RK) methods. A s-Runge-Kutta method for the solution of the IVP (4.1) can be written as

$$\mathbf{k}_i = \mathbf{f}(t_n + c_i \Delta t_n, \underline{\mathbf{u}}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j), \quad i = 1, ..., s \tag{4.14}$$

$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^n + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i,$$

where the vectors $\mathbf{k}_j$ are called stage derivatives. For the autonomous system (4.2), this simplifies to

$$\mathbf{k}_i = \mathbf{f}(\underline{\mathbf{u}}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j), \quad i = 1, ..., s, \tag{4.15}$$

$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^n + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i.$$

The coefficients of the scheme can be written in a compact form using the vectors $\mathbf{c} = (c_1, ..., c_s)^T$, $\mathbf{b} = (b_1, ..., b_s)^T$ and the matrix $\mathbf{A} = (a_{ij})_{ij} \in \mathbb{R}^{s\times s}$. Typically, these are then combined in the so called Butcher array, which is the standard way of writing down the coefficients of an RK scheme:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \cdots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} = \begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}$$

An explicit Runge-Kutta method is obtained, if the matrix $\mathbf{A}$ is strictly lower triangular, otherwise the method is implicit. The stability function is the rational function

$$R(z) = \frac{\det(\mathbf{I} - z\mathbf{A} + z\mathbf{e}\mathbf{b}^T)}{\det(\mathbf{I} - z\mathbf{A})}, \tag{4.16}$$

where $\mathbf{e} \in \mathbb{R}^s$ is the vector of all ones. This can be seen by looking at the equation system obtained when solving the linear test equation and solving that using Cramer's rule.

The maximal order of an s-stage Runge-Kutta method is $2s$, which is obtained by the so called Gauß-Legendre methods. One way to obtain Runge-Kutta methods of a certain order is to use Taylor expansions and coefficient matching. To keep track of the high order derivatives, the concept of Butcher trees is used. Nevertheless, this results in a huge number of nonlinear equations. Therefore, Butcher devised simplifying conditions that allow to solve these nonlinear systems even for high orders. In this way, it is possible to find Runge-Kutta methods with different orders, degrees of freedom and stability properties.

Several time integration schemes are widely used, for example the Crank-Nicholson scheme, also called trapezoidal rule, which is A-stable. Another example is the implicit Midpoint rule, which is of second order and also A-stable.

4.5.1 Explicit Runge-Kutta methods

There is a variety of popular explicit Runge-Kutta methods, in particular the explicit Euler method, the improved Euler method, which is a second order explicit Runge-Kutta scheme or the classical Runge-Kutta method of fourth order. Regarding stability, the stability function (4.7) of an s-stage explicit Runge-Kutta method is a polynomial of degree s:

$$R(z) = \sum_{i=0}^{s} \alpha_i z^i. \tag{4.17}$$

Therefore the stability region is bounded and thus, these methods cannot be A-stable.

Regarding order, since a large number of the a_{ij} is zero, there are significantly less degrees of freedom in the design of the method. Therefore, the order of an s-stage method can be up to s for $s \leq 4$. Beyond that, the maximal order of an s-stage explicit RK method is smaller than s.

As for the choice of method, explicit RK methods are useful if the stability constraint is not severe with regards to the time scale we are interested in. Now, if oscillations are a problem, SSP methods should be employed. This is essentially never the case for finite volume methods, where the inherent diffusion is quite large. However, for high order methods like DG, oscillations can become a problem. The coefficients of k-stage, kth-order methods with an optimal SSP constant of 1 can be found in table (B.1).

If oscillations are not an issue, low storage explicit RK method are a good option, that need only two vectors for the solution. These are given by

$$\begin{aligned}
&\underline{\mathbf{u}} = \underline{\mathbf{u}}^n \\
&\underline{\mathbf{k}} = \underline{\mathbf{0}} \\
&i \in [1, ..., s] : \begin{cases} \underline{\mathbf{k}} = a_i \ \underline{\mathbf{k}} + \Delta t \mathbf{f}\left(\underline{\mathbf{u}}\right) \\ \underline{\mathbf{u}} = \underline{\mathbf{u}} + b_i \ \underline{\mathbf{k}} \end{cases} \\
&\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}
\end{aligned} \tag{4.18}$$

Suitable coefficients can be found in the appendix, table B.3.

4.5.2 DIRK methods

In contrast to multistep schemes, no bound on the order for A-stable schemes has been proved or observed for implicit Runge-Kutta methods. This is because the stability function is rational and therefore, the stability region can be unbounded. However, a general implicit Runge-Kutta method requires solving a nonlinear equation system with $s \cdot m$ unknowns. Several methods have been suggested to avoid this, for example SIRK methods where $\mathbf{A}$ is diagonalizable. We will restrict ourselves to so called diagonally implicit Runge-Kutta methods or short DIRK methods. Given coefficients a_{ij} and b_i, such a method with s stages can be written as

$$\mathbf{k}_i = \underline{\mathbf{f}}(t_n + c_i\Delta t, \underline{\mathbf{u}}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j), \quad i = 1, ..., s \tag{4.19}$$
$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^n + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i.$$

Thus, all entries of the Butcher array in the strictly upper triangular part are zero. If additionally all values on the diagonal of $\mathbf{A}$ are identical, the scheme is called singly diagonally implicit, short SDIRK. The stability function of a DIRK method is given by

$$R(z) = \frac{\det(\mathbf{I} - z\mathbf{A} + z\mathbf{e}\mathbf{b}^T)}{\Pi_{i=1}^{s}(1 - za_{ii})}.$$

Regarding order, an s-stage SDIRK method can be of order $s+1$. This is a bit surprising, since there is only one additional degree of freedom compared to an explicit method. A way to obtain an L-stable scheme is to require, that the coefficients $\mathbf{b}$ and the last line of the matrix $\mathbf{A}$ coincide. This class of schemes is called stiffly accurate and is popular for the solution of differential algebraic equations (DAEs). Unfortunately, but not unexpectedly, these schemes have at most order s. Finally, there is the class of ESDIRK schemes, where the first step of the Runge-Kutta schemes is explicit. The butcher arrays of SDIRK and ESDIRK methods are illustrated in table 4.1.

$$\begin{array}{c|cccc} \alpha & \alpha & 0 & 0 & 0 \\ c_2 & a_{21} & \alpha & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ c_s & a_{s1} & \dots & a_{ss-1} & \alpha \\ \hline & b_1 & \dots & \dots & b_s \end{array} \qquad \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ c_2 & a_{21} & \alpha & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ c_s & a_{s1} & \dots & a_{ss-1} & \alpha \\ \hline & b_1 & \dots & \dots & b_s \end{array}$$

Table 4.1: Butcher array of a DIRK method (left) and an ESDIRK method

The point about DIRK schemes is, that the computation of the stage vectors is decoupled and instead of solving one nonlinear system with sm unknowns, the s nonlinear systems (4.19) with m unknowns have to be solved. This corresponds to the sequential application of several implicit Euler steps in two possible ways. With the starting vectors

$$\mathbf{s}_i = \underline{\mathbf{u}}^n + \Delta t \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j, \tag{4.20}$$

we can solve for the stage derivatives

$$\mathbf{k}_i = \underline{\mathbf{f}}(\mathbf{s}_i + \Delta t a_{ii}\mathbf{k}_i) \tag{4.21}$$

or we define the stage values via

$$\mathbf{U}_i = \underline{\mathbf{u}}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j, \tag{4.22}$$

which implies

$$\mathbf{k}_i = \underline{\mathbf{f}}(\mathbf{U}_i)$$

and then solve the equation

$$\mathbf{U}_i = \mathbf{s}_i + \Delta t a_{ii}\underline{\mathbf{f}}(\mathbf{U}_i). \tag{4.23}$$

In the autonomous case, the equations (4.21) and (4.23) correspond to one step of the implicit Euler method with starting vector $\mathbf{s}_i$ and time step $a_{ii}\Delta t$. If (4.23) is employed, the stage value $\mathbf{k}_i$ is then obtained via

$$\mathbf{k}_i = (\mathbf{U}_i - \mathbf{s}_i)/(a_{ii}\Delta t),$$

which avoids a costly and for stiff problems error prone evaluation of the right hand side [174]. The major difference between the two formulations is that the iterative solver used to solve one of the equations will produce errors in either the stage values or the stage derivatives, leading to a different error propagation [151]. Finally, we apply the formula

$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^n + \Delta t \sum_{i=1}^{s} b_i\mathbf{k}_i.$$

Note that this is not necessary for stiffly accurate methods that work with stage values, since there $\underline{\mathbf{u}}^{n+1} = \mathbf{U}_s$. Therefore, we will only consider stiffly accurate methods, since they have beneficial stability properties and less computational costs.

All in all, the application of a DIRK scheme corresponds to the solution of s nonlinear systems of the form (4.19) per time step and therefore, the intuitive thought is not to consider this class of schemes, since they are inefficient. However, several facts lead to a positive reevaluation of these schemes. First of all, a_{ii} is typically smaller than one and thus the equation systems are easier to solve than a system arising from an implicit Euler discretization with the same Δt. Second, if a time adaptive strategy is used, the higher order leads to larger time steps compared to the implicit Euler method and therefore fewer time steps are needed to integrate from t_0 to t_{end}, leading to a smaller total number of nonlinear systems to be solved. Finally, since we have a sequence of nonlinear systems which continuously depend on each other, results from previous systems can be exploited to speed up the solution of later systems. This will be explained in the following chapter.

Additionally, it is usually possible to give a second set of coefficients $\hat{b}$, which define for the otherwise identical Butcher tableau a method of lower order. This can be used for the estimation of the time integration error, as will be explained later. Furthermore, so called dense output formulas can be found. These are designed to deliver values of the solution inside the interval $[t_n, t_{n+1}]$, once $\underline{\mathbf{u}}^{n+1}$ has been computed. However, they can be used to extrapolate the solution into the future to obtain starting values for the iterative solution methods for the nonlinear equation systems. This will be discussed in the next chapter. The formulas itself are given by

$$\underline{\mathbf{u}}(t_n + \theta \Delta t) = \underline{\mathbf{u}}^n + \Delta t \sum_{i=1}^{s} b_i^*(\theta) \mathbf{f}(\underline{\mathbf{u}}^{(i)}), \tag{4.24}$$

where the factors $b_i^*(\Theta)$ are given by

$$b_i^*(\theta) = \sum_{j=1}^{p^*} b_{ij}^* \theta^j, \quad b_i^*(\theta = 1) = b_i, \tag{4.25}$$

with b_{ij}^* being a set of coefficients and p^* the order of the formula.

One example for an SDIRK scheme is the method of Ellsiepen [57], which is of second order with an embedded method of first order, see table (B.4). A method of third order with an embedding of second order was developed by Cash, see table (B.5). In the context of solid mechanics, it has been demonstrated by Hartmann that these methods are competitive [85].

Regarding ESDIRK schemes, this allows to have a stage order of two, but also means that the methods cannot be algebraically stable. The use of these schemes in the context of compressible Navier-Stokes equations was analyzed by Bijl et al. in [17] where they were demonstrated to be more efficient than BDF methods for engineering accuracies. They suggested the six stage method ESDIRK4 of fourth order with an embedded method of third order and the four stage method ESDIRK3 of third order with an embedded method of second order, both designed in [109]. The coefficients can be obtained from the diagrams (B.6) and (B.7), respectively (B.8) and (B.9) for the dense output formulas. This result about the comparative efficiency of BDF-2 and ESDIRK4 was later confirmed by Jothiprasad et. al. for finite volume schemes [103] and Wang and Mavriplis for a DG discretization of unsteady Euler flow [219].

The use of a first explicit stage $\mathbf{k}_1 = \underline{\mathbf{f}}(\underline{\mathbf{u}}^n)$ in a stiffly accurate method allows to reuse the final stage derivative $\mathbf{k}_s$ from the last time step in the first stage, since this is equal to $\underline{\mathbf{f}}(\underline{\mathbf{u}}^{n-1} + \Delta t_{n-1} \sum_{i=1}^{s} b_i \mathbf{k}_i)$. Besides saving a modest amount of computing time, this is in particular advisable for stiff problems with a large Lipschitz constant. There, small errors in the computation of stage values may lead to large errors in evaluated functions, whereas small errors in the computed stage derivative are just that. The drawback here is that the methods then no longer are onestep methods in the strict sense, since we have to store the last stage derivative.

4.5.3 Additive Runge-Kutta methods

An idea that came up in the 1980s is to use different Runge-Kutta methods for different terms in the equation, which was introduced to the CFD community by Jameson. He used explicit additive methods as smoothers in his multigrid methods and found that it was beneficial to treat the convective and diffusive terms differently. This will be explained later. Another point about this are implicit-explicit methods (IMEX) as designed in [109], which treat nonstiff terms explicitly and stiff terms implicitly. We will here represent the idea for two terms, but the extension to N terms is straightforward. Consider the ODE

$$\frac{d}{dt}\underline{\mathbf{u}}(t) = \underline{\mathbf{f}}(t, \underline{\mathbf{u}}(t)), \quad \underline{\mathbf{u}}(t_0) = \underline{\mathbf{u}}_0, \quad t \in [t_0, t_{end}], \tag{4.26}$$

with

$$\underline{\mathbf{f}}(t, \underline{\mathbf{u}}(t)) = \underline{\mathbf{f}}_1(t, \underline{\mathbf{u}}(t)) + \underline{\mathbf{f}}_2(t, \underline{\mathbf{u}}(t)).$$

The method is then applied as

$$\begin{aligned} \mathbf{U}_i &= \underline{\mathbf{u}}^n + \Delta t \sum_{\nu=1}^{2} \sum_{j=1}^{i} a_{ij}^{(\nu)} \underline{\mathbf{f}}(\mathbf{U}_j), \quad i = 1, ..., s \qquad (4.27) \\ \underline{\mathbf{u}}^{n+1} &= \underline{\mathbf{u}}^n + \Delta t \sum_{\nu=1}^{2} \sum_{i=1}^{s} b_i^{(\nu)} \underline{\mathbf{f}}(\mathbf{U}_i). \end{aligned}$$

4.6 Rosenbrock-type methods

To circumvent the solution of nonlinear equation systems, so called Rosenbrock methods can be used, also referred to as Rosenbrock-Wanner or short ROW methods, see [82] or the german volume [190]. The idea is to linearize an s-stage DIRK scheme, thus sacrificing some stability properties, as well as accuracy, but reducing the computational effort in that per time step, s linear equation systems with the same system matrix and different right hand sides have to be solved. Therefore, this class of schemes is also sometimes referred to as linearly implicit or semi-implicit. Note that the last term is used in an ambiguous way in different contexts and therefore shouldn't be used. In [102], Rosenbrock methods are compared to SDIRK methods in the context of the incompressible Navier-Stokes equations and found to be competitive, if not superior. The use of Rosenbrock methods in the context of compressible flow problems is so far quite rare. One example is the work of St.-Cyr et. al. in the context of DG [186].

For the derivation of the schemes we start by linearizing formula (4.19) around $\mathbf{s}_i$ (4.20) to obtain

$$\mathbf{k}_i \approx \underline{\mathbf{f}}(\mathbf{s}_i) + \Delta t a_{ii} \frac{\partial \underline{\mathbf{f}}(\mathbf{s}_i)}{\partial \underline{\mathbf{u}}} \mathbf{k}_i.$$

To avoid a recomputation of the Jacobian, we replace $\frac{\partial \underline{\mathbf{f}}(\mathbf{s}_i)}{\partial \underline{\mathbf{u}}}$ by $\mathbf{J} = \frac{\partial \underline{\mathbf{f}}(\underline{\mathbf{u}}^n)}{\partial \underline{\mathbf{u}}}$. Finally, to gain more freedom in the definition of the method, linear combinations of $\Delta t \mathbf{J} \mathbf{k}_i$ are added to the last term. Since the linearization procedure can be interpreted as performing just one Newton step at every stage of the DIRK method instead of a Newton loop, the added terms correspond roughly to choosing that as the initial guess for the first Newton iteration. If instead of the exact Jacobian, an approximation $\mathbf{W} \approx \mathbf{J}$ is used, we obtain so called W-methods. Since for the systems considered here, exact solutions are impossible, we will consider W-methods only. If the linear system is solved using a Krylov subspace method (see section 5.8), the scheme is called a Krylov-ROW method. This is for example implemented in the code ROWMAP [220].

We thus obtain an s-stage Rosenbrock-Wanner method with coefficients a_{ij}, γ_{ij} and b_i in the form

$$\begin{aligned}(\mathbf{I} - \gamma_{ii}\Delta t\mathbf{W})\mathbf{k}_i &= \underline{\mathbf{f}}(\mathbf{s}_i) + \Delta t\mathbf{W}\sum_{j=1}^{i-1}\gamma_{ij}\mathbf{k}_j, \quad i = 1, ..., s \\ \mathbf{s}_i &= \underline{\mathbf{u}}^n + \Delta t\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j, \quad i = 1, ..., s \\ \underline{\mathbf{u}}^{n+1} &= \underline{\mathbf{u}}^n + \Delta t\sum_{i=1}^{s} b_i\mathbf{k}_i.\end{aligned} \tag{4.28}$$

Here, the coefficients a_{ij} and b_i correspond to those of the DIRK method and the γ_{ii} are the diagonal coefficients of that, whereas the offdiagonal γ_{ij} are additional coefficients.

In the case of a nonautonomous equation (4.1), we obtain an additional term $\Delta t\gamma_i\partial_t\mathbf{f}(t_n, \underline{\mathbf{u}}_n)$ with $\gamma_i = \sum_{j=1}^{i}\gamma_{ij}$ on the right hand side of (4.28) and thus:

$$\begin{aligned}(\mathbf{I} - \gamma_{ii}\Delta t\mathbf{W})\mathbf{k}_i &= \underline{\mathbf{f}}(t_n + a_i\Delta t, \mathbf{s}_i) + \Delta t\mathbf{W}\sum_{j=1}^{i-1}\gamma_{ij}\mathbf{k}_j + \Delta t\gamma_i\partial_t\underline{\mathbf{f}}(t_n, \underline{\mathbf{u}}_n), \quad i = 1, ..., s \\ \mathbf{s}_i &= \underline{\mathbf{u}}^n + \Delta t\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j, \quad i = 1, ..., s \\ \underline{\mathbf{u}}^{n+1} &= \underline{\mathbf{u}}^n + \Delta t\sum_{i=1}^{s} b_i\mathbf{k}_i,\end{aligned} \tag{4.29}$$

with $a_i = \sum_{j=1}^{i-1} a_{ij}$.

Regarding order, order trees for Rosenbrock methods can be derived using the same techniques as for Runge-Kutta methods and the result is very similar to that for DIRK methods. For W-methods, additional order conditions are needed to avoid a loss of order.

Generally speaking, Rosenbrock methods are less accurate than SDIRK methods, which will result later in the time step selector chosing smaller time steps for Rosenbrock methods. Regarding stability, the stability function of a Rosenbrock method is that of a DIRK method where the matrix in the butcher array has entries $a_{ij} + \gamma_{ij}$. It turns out that it is possible to design A-stable and even L-stable W-methods.

The efficient implementation of Rosenbrock methods is done using a set of auxiliary variables

$$\mathbf{U}_i = \Delta t \sum_{j=1}^{i} \gamma_{ij} \mathbf{k}_j$$

and thus circumvents the matrix-vector multiplication in the previous formulation (4.28). Using the identity

$$\mathbf{k}_i = \frac{1}{\Delta t}\left(\frac{1}{\gamma_{ii}}\mathbf{U}_i - \sum_{j=1}^{i-1} c_{ij}\mathbf{U}_j\right)$$

with coefficients c_{ij} explained below, we obtain

$$\begin{aligned}
(\mathbf{I} - \gamma_{ii}\Delta t \mathbf{W})\mathbf{U}_i &= \Delta t \gamma_{ii} \underline{\mathbf{f}}(\hat{\mathbf{s}}_i) + \gamma_{ii}\sum_{j=1}^{i-1} c_{ij}\mathbf{U}_j, \quad i = 1, ..., s, \qquad (4.30)\\
\hat{\mathbf{s}}_i &= \underline{\mathbf{u}}^n + \sum_{j=1}^{i-1} \tilde{a}_{ij}\mathbf{U}_j, \quad i = 1, ..., s,\\
\underline{\mathbf{u}}^{n+1} &= \underline{\mathbf{u}}^n + \sum_{i=1}^{s} m_i \mathbf{U}_i.
\end{aligned}$$

The relation between the two sets of coefficients is the following:

$$\mathbf{C} = \operatorname{diag}(\gamma_{11}^{-1}, ..., \gamma_{ss}^{-1}) - \mathbf{\Gamma}^{-1}, \qquad \tilde{\mathbf{A}} = \mathbf{A}\mathbf{\Gamma}^{-1}, \qquad \mathbf{m}^T = \mathbf{b}^T\mathbf{\Gamma}^{-1},$$

where $\Gamma = (\gamma_{ij})_{ij}$. Again, in the nonautonomous case, we obtain an additional term on the right hand side and thus:

$$\begin{aligned}
(\mathbf{I} - \gamma_{ii}\Delta t \mathbf{W})\mathbf{U}_i &= \Delta t \gamma_{ii} \underline{\mathbf{f}}(t_n + a_i\Delta t, \hat{\mathbf{s}}_i) + \gamma_{ii}\sum_{j=1}^{i-1} c_{ij}\mathbf{U}_j + \Delta t^2 \gamma_{ii}\gamma_i \partial_t \mathbf{f}(t_n, \underline{\mathbf{u}}_n), \quad i = 1, ..., s, \qquad (4.31)\\
\hat{\mathbf{s}}_i &= \underline{\mathbf{u}}^n + \sum_{j=1}^{i-1} \tilde{a}_{ij}\mathbf{U}_j, \quad i = 1, ..., s,\\
\underline{\mathbf{u}}^{n+1} &= \underline{\mathbf{u}}^n + \sum_{i=1}^{s} m_i \mathbf{U}_i.
\end{aligned}$$

When coding a Rosenbrock method from a given set of coefficients, extra care should be taken, since different authors define the coefficients in slightly different ways, including different scalings by Δt in the definition of the method. Further possible confusion arises from the two different formulations. Here, the coefficients for both formulations will always be given.

As in the case of DIRK methods, the fact that we have a sequence of linear systems, can be exploited. In particular, we have a sequence of s linear systems with the same matrix, but varying right hand side, following in the next time step by a sequence of another s linear systems, with a new matrix that is not arbitrary far away from the last. Approaches for this are described in the next chapter.

One example for a Rosenbrock method is ROS34PW2, which was introduced in [161]. This method has four stages and is of order 3 with an embedded method of order 2. It is a W-method and L-stable. The Butcher array is given in table (B.10). Another example of an L-stable W-method is given by the 6-stage, 4th order method RODASP, which can be seen in table (B.11) and has an embedded method of order 3.

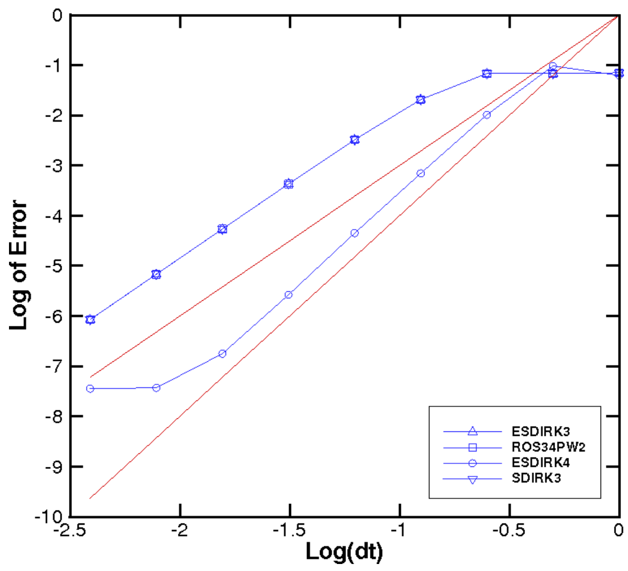


Figure 4.4: Order of different time integration methods in the DG case.

A comparison of the order and error of different implicit time integration schemes can be seen in figure 4.4, where a DG discretization is used for a weakly compressible flow problem. Due to the weak nonlinearity, there is little difference in the errors of Rosenbrock and DIRK schemes.

4.7 Adaptive time step size selection

For an explicit method, the CFL condition is typically so strict that when choosing the time step based on it, the time integration error is below the error tolerance. This is not the case for an implicit method, which means that there, a different strategy is needed for the time step selection. Often, a CFL number is stated and the time step computed accordingly. This has a number of drawbacks. First of all, while the CFL number corresponds to a well defined time step on a specific grid for a specific solution, its size is not obvious, whereas for unsteady computations, we often think in time scales based on physical time. Second, when we assume that the CFL number has been chosen such that the time integration error is reasonable, this might be inefficient if much larger time steps would be possible. Finally, even if typically the time integration error is significantly smaller than the space discretization error, we would still like to know the size of the time integration error or otherwise put, we want to control it.

Therefore, time adaptive strategies that choose the time step based on estimates of the time integration error are to be preferred. For DIRK and Rosenbrock methods, this is done using the embedded schemes of a lower order $\hat{p}$. Comparing the local truncation error of both schemes, we obtain an estimate of the local error $\hat{\mathbf{l}}$ of the lower order scheme:

$$\hat{\mathbf{l}} \approx \Delta t_n \sum_{j=1}^{s} (b_i - \hat{b}_i)\mathbf{k}_i, \tag{4.32}$$

respectively, for the efficient formulation (4.31) of the Rosenbrock method

$$\hat{\mathbf{l}} \approx \sum_{j=1}^{s} (m_i - \hat{m}_i)\mathbf{U}_i. \tag{4.33}$$

For BDF methods, the error estimate is obtained from taking a two steps with half the time step size, resulting in $\underline{\mathbf{u}}_2$ and using Richardson extrapolation. The local error $\mathbf{l}_2$ after the two half steps for a method of order p is estimated as

$$\mathbf{l}_2 \approx \frac{\underline{\mathbf{u}}_2 - \underline{\mathbf{u}}^{n+1}}{2^p - 1}. \tag{4.34}$$

The local error estimate is then used to determine the new step size. To do this, we decide beforehand on a target error tolerance, which we implement using a common fixed resolution test [181]. This means that we define the error tolerance per component via

$$d_i = RTOL|u_i^n| + ATOL. \tag{4.35}$$

Typically, we choose $RTOL = ATOL$, so that there is only one input parameter for this. We then compare (4.35) to the local error estimate via requiring

$$\|\hat{\mathbf{l}}./\mathbf{d}\| \leq 1, \tag{4.36}$$

where the . denotes a pointwise division operator. An important question here is the choice of norm. First of all, the maximum norm guarantees that the tolerance test (4.35) is satisfied in every cell. However, the 2-Norm is typically more accurate overall and since we have to define a tolerance for the subsolver, namely the Newton-GMRES method in some norm, it is natural to use this one, since GMRES uses a 2-norm minimization. Therefore, we will use the latter, since it seems to work well in practice. Nevertheless a more substantial and less heuristic choice of norm would be desirable for the future.

The next question is, how the time step has to be chosen, such that the error can be controlled. Under the assumption that higher order terms can be neglected in a Taylor expansion of the local error, we have

$$\|\hat{\mathbf{l}}\| \approx \|\Delta t_n^{\hat{p}+1}\hat{\Psi}(t_n, \underline{\mathbf{u}})\| \approx \|\underline{\hat{\mathbf{u}}}^{n+1} - \underline{\mathbf{u}}^{n+1}\|,$$

where $\hat{\Psi}(t_n, \underline{\mathbf{u}})$ is the error function of the embedded method. If we additionally assume that $\hat{\Psi}(t, \underline{\mathbf{u}})$ is slowly varying, we obtain the classical method, see [82]:

$$\Delta t_{new} = \Delta t_n \cdot \|\mathbf{l}./\mathbf{d}\|^{-1/k}. \tag{4.37}$$

Here, $k = \hat{p}$ leads to a so called EPUS (error per unit step) control, whereas $k = \hat{p}+1$ is an EPS (error per step) control. We will use EPS, which often results in more efficient schemes. Formula (4.37) is combined with safety factors to avoid volatile increases or decreases in time step size:

$$\begin{aligned} \text{if } \|\mathbf{l}./\mathbf{d}\| \geq 1, \qquad & \Delta t_{n+1} = \Delta t_n \max(f_{min}, f_{safety}\|\mathbf{l}./\mathbf{d}\|^{-1/\hat{p}+1}) \\ \text{else} \qquad & \Delta t_{n+1} = \Delta t_n \min(f_{max}, f_{safety}\|\mathbf{l}./\mathbf{d}\|^{-1/\hat{p}+1}). \end{aligned}$$

The classical controller is what corresponds to the I-controller in control theory. More elaborate controllers that take the step size history into account are possible, for example the PID controller [179]. Söderlind suggested a controller using methods from signal processing [178].

Finally, tolerance scaling and calibration should be used [180]. Tolerance scaling is useful, since, although for the above controller, convergence of the method for $TOL \to 0$ can be proven, the relation between global error and tolerance is typically of the form

$$\|e\| = \tau \cdot TOL^{\alpha},$$

where α is smaller than one, but does not depend strongly on the problem solved and τ is a proportionality factor. Therefore, an internal rescaling $TOL' = \mathcal{O}(TOL^{1/\alpha})$ should be done, so that the user obtains a behavior where a decrease in TOL leads to a corresponding decrease in error.

Tolerance calibration is used to make sure that at least for a reference problem and a specific tolerance TOL_0, TOL and TOL' are the same. This is achieved via the rescaling

$$TOL' = TOL_0^{(\alpha-1/\alpha)}TOL^{1/\alpha}. \tag{4.38}$$

Through numerical experiments, we suggest the values in table 4.7 for α for different methods [24].

4.8 Operator Splittings

Consider an initial value problem with a right hand side split in a sum of two terms $\underline{\mathbf{f}}_1$ and $\underline{\mathbf{f}}_2$, arising from the discretization of a balance law

	SDIRK 3	ESDIRK 3	ESDIRK 4	ROS34PW2
α	0.9	0.9	0.9	0.8

Table 4.2: Values for α chosen for different time integration methods

$$\frac{d}{dt}\underline{\mathbf{u}}(t) = \underline{\mathbf{f}}_1(t, \underline{\mathbf{u}}(t)) + \underline{\mathbf{f}}_2(t, \underline{\mathbf{u}}(t)), \quad \underline{\mathbf{u}}(t_0) = \underline{\mathbf{u}}^0, \quad t \in [t_0, t_{end}]. \tag{4.39}$$

For example, the splitting could correspond to the x and y dimension in the Navier-Stokes equations, $\underline{\mathbf{f}}_1$ could be the discretization of the inviscid fluxes, whereas $\underline{\mathbf{f}}_2$ is the discretization of the viscous fluxes. Or $\underline{\mathbf{f}}_1$ could be both viscous and inviscid fluxes, whereas $\underline{\mathbf{f}}_2$ arises from the discretization of a source term like gravity.

The terms $\underline{\mathbf{f}}_1$ and $\underline{\mathbf{f}}_2$ interact and the most accurate way to treat these terms is to respect this interaction in the spatial discretization. This was discussed in section 3.6. However, it is sometimes easier to treat these terms separately. This is called an *operator splitting* or *fractional step* method. These split the solution process into the solution of two ordinary differential equations, which is useful if for both equations, well known methods are available. A first order approximation (for smooth solutions $\mathbf{u}$) to this problem is given by the simple Godunov splitting [70]:

1. Solve $\frac{d}{dt}\mathbf{u} + \underline{\mathbf{f}}_1(\mathbf{u}) = \mathbf{0}$ with timestep Δt and initial data $\mathbf{u}^n$ to obtain intermediate data $\mathbf{u}^*$.
2. Solve $\frac{d}{dt}\mathbf{u} + \underline{\mathbf{f}}_2(\mathbf{u}) = \mathbf{0}$ with the same timestep, but inital data $\mathbf{u}^*$ to obtain $\mathbf{u}^{n+1}$.

Here, we require each "solve" to be at least first order accurate.

To increase the accuracy and obtain a scheme of second order for smooth solutions, we have to use a slightly more sophisticated method, for example the Strang splitting [188], where again the subproblem solvers have to be at least of first order:

1. Solve $\frac{d}{dt}\mathbf{u} + \underline{\mathbf{f}}_1(\mathbf{u}) = \mathbf{0}$ with timestep $\Delta t/2$.
2. Solve $\frac{d}{dt}\mathbf{u} + \underline{\mathbf{f}}_2(\mathbf{u}) = \mathbf{0}$ with timestep Δt.
3. Solve $\frac{d}{dt}\mathbf{u} + \underline{\mathbf{f}}_1(\mathbf{u}) = \mathbf{0}$ with timestep $\Delta t/2$.

As before, in each step, the intermediate result obtained in the last step is used as inital data. Since the problem with $\underline{\mathbf{f}}_1$ is solved twice in this operator splitting, $\underline{\mathbf{f}}_1$ should be chosen such that it is easier to solve than the problem with $\underline{\mathbf{f}}_2$. Splittings of order greater than two can only be defined under severe assumptions on the operators [30].

The role of $\underline{\mathbf{f}}_1$ and $\underline{\mathbf{f}}_2$ can of course be exchanged, however in general they do not commute. This becomes obvious when considering a local heat source. Increasing the heat first and then applying the convective flux leads to a different result compared to doing the convective step first and then increasing the heat locally. For this reason, special care has to be taken in chosing the numerical boundary conditions for the partial differential equation. Otherwise, unphysical effects can be introduced into the solution.

The ”solves” in each step correspond to a time integration procedure, which has to be chosen of the appropriate order. For source term, the idea is that for these, typically quite simple time integration procedures can be chosen, possibly leading to more efficient overall schemes than when incorporating the source term into the computation of the fluxes. For the Navier-Stokes equations, an important idea is to use an operator splitting where an implicit time integration method is used for the diffusive fluxes and an explicit method is used for the convective parts, since the CFL condition is typically less severe than the DFL condition. This is sometimes referred to as an IMEX scheme for implicit-explicit, but care has to be taken since this term is also used for schemes where an explicit or implicit scheme is used depending on the part of the spatial domain considered.

Tang and Teng [193] proved for multidimensional scalar balance laws that if the exact solution operator is used for both subproblems, the described schemes converge to the weak entropy solution and furthermore that the L^1 convergence rate of both fractional step methods is not worse than 1/2. This convergence rate is actually optimal, if a monotone scheme is used for the homogenous conservation law in combination with the forward Euler method for the time integration. Langseth, Tveito and Winther [121] proved for scalar one dimensional balance laws that the L^1 convergence rate of the Godunov splitting (again using the exact solution operators) is linear and showed corresponding numerical examples, even for systems of equations. A better convergence rate than linear for nonsmooth solutions is not possible, as Crandall and Majda proved already in 1980 [42].

The L^1 error does not tell the whole story. Using the Strang or Godunov splitting combined with a higher order method in space and a second order time integration does improve the solution compared with first order schemes and is therefore appropriate for the computation of unsteady flows. This is for example suggested by LeVeque [124].

Regarding time adaptivity, embedded Runge-Kutta methods cannot be used and Richardson extrapolation has to be used instead.

4.9 Alternatives to the method of lines

So far, we have looked at the method of lines only. In [130], Mani and Mavriplis follow a different approach in the FV case in that they write down a huge equation system for all the unknowns at several time steps simultanously and combine this with a time adaptive strategy. An alternative to discretizing in space first and then in time, is to discretize in space and time simultaneously. This approach is unusual in the finite volume context, but followed for example in the ADER method [197]. For DG methods, this is slightly more common, for example in the space-time DG of Klaij et. al. [111] or the space-time expansion (STE) DG of Lörcher et. al. [128], which allows for a local time stepping via a predictor-corrector scheme to increase efficiency for unsteady flows. There and in other approaches, the time integration over the interval $[t_n, t_{n+1}]$ is embedded in the overall DG formulation.

4.9.1 Local time stepping Predictor-Corrector-DG

As an example of an alternative method, we will now explain the Predictor-Corrector-DG method and the local time stepping used in more detail. Starting point is the evolution equation (3.30) for the cell i, integrated over the time interval $[t_n, t_{n+1}]$:

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n - \int_{t_n}^{t_{n+1}} \underbrace{\sum_{i=1}^{nFaces} \mathbf{M}_i^S \mathbf{g_i}}_{\mathbf{R}_V(p_i)} - \underbrace{\sum_{k=1}^{d} \mathbf{S}_k \mathbf{f}_k}_{\mathbf{R}_S(p_i,p_j)} \, dt.$$

The time integral is approximated using Gaussian quadrature. This raises the question of how to obtain the values at future times. To this end, the integral is split into the volume term defined by $\mathbf{R}_V$ that needs information from the cell i only and the surface term $\mathbf{R}_S$ that requires information from neighboring cells. Then, the use of cellwise predictor polynomials $\mathbf{p}_i(t)$ in time is suggested. Once this is given, the update can be computed via

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n - \int_{t_n}^{t_{n+1}} \mathbf{R}_V(\mathbf{p}_i) - \mathbf{R}_S(\mathbf{p}_i, \mathbf{p}_j) dt. \quad (4.40)$$

Several methods to obtain these predictor polynomials have been suggested. The first idea was to use a space time expansion via the Cauchy-Kowalewskaja procedure [128], which leads to a rather costly and cumbersome scheme. However, in [62], the use of continuous extension Runge-Kutta (CERK) methods is suggested instead. This is a type of RK methods that allows to obtain approximations to the solution not only at t_{n+1}, but at any value $t \in [t_n, t_{n+1}]$ [153]. The only difference to the dense output formulas (4.24) and (4.25) mentioned earlier is that the latter are designed for particular RK methods, whereas the CERK methods are full RK schemes in their own right. Coefficients of an explicit four stage CERK method with its continuous extension can be found in the appendix in tables B.12 and B.13.

The CERK method is then used to integrate the initial value problem

$$\frac{d}{dt}\mathbf{u}_i(t) = \mathbf{R}_V(\mathbf{u}_i), \quad \mathbf{u}_i(t_n) = \mathbf{u}_i^n, \quad (4.41)$$

in every cell i, resulting in stage derivatives $\mathbf{k}_j$. The values at the Gauss points are obtaind via the CERK polynial,

$$\mathbf{p}(t) = \sum_{k=0}^{p} \mathbf{q}_k t^k$$

which is of order p corresponding to the order of the CERK method minus one and has the coefficients

$$\mathbf{q}_k = \frac{1}{\Delta t_n^k} \sum_{j=1}^{s} b_{kj} \mathbf{k}_j,$$

where the coefficients b_{kj} can be found in table B.13.

The predictor method needs to be of order $k-1$, if order k is sought for the complete time integration. If a global time step is employed, the method described so far is already applicable. However, a crucial property of the scheme is that a local time stepping procedure can be used.

A cell i can be advanced in time, if the necessary information in the neighboring cells is already there, thus if the local time at time level n is not larger than the time level in the neighboring cells:

$$t_i^{n+1} \leq \min \left\{ t_j^{n+1} \right\} \forall j \in N(i). \tag{4.42}$$

This is illustrated in figure 4.5. There, all cells are synchronized at time level t_n, then in each cell a predictor polynomial is computed using the CERK method, which is valid for the duration of the local time step Δt_{n_i}. However, for the completion of a time step in both cell $i-1$ and $i+1$, boundary data is missing. However, cell i fulfills the evolve condition and thus, the time step there can be completed using the predictor polynomials p_{i-1} and p_{i+1}. Then, the predictor polynomial in cell i for the next local time step is computed. Now, cell $i-1$ fulfills the evolve condition and after the completion of that time step, the second time step in cell i can be computed. Note that while this example looks sequential, on a large grid, a number of cells can be advanced in time in parallel, making the scheme attractive on modern architectures.

Finally, to make the scheme conservative, care needs to be taken when a cell is advanced in time, whose neighbor has been partially advanced. Thus, the appropriate time integral is split into two parts, which are computed separately:

$$\int_{t^n}^{t^{n+1}} ...dt = \int_{t^n}^{t^1} ...dt + \int_{t^1}^{t^2} ...dt + ... + \int_{t^{K-2}}^{t^{K-2}} ...dt + \int_{t^{K-1}}^{t^{n+1}} ...dt,$$

we split the interval $\left[t_1^n, t_1^{n+1}\right]$ into the intervals $\left[t_1^n, t_2^{n+1}\right]$ and $\left[t_2^{n+1}, t_1^{n+1}\right]$ which yields

$$\int_{t_1^n}^{t_1^{n+1}} \mathbf{R}_S\left(\tilde{p}_1, \tilde{p}_2\right) dt = \int_{t_1^n}^{t_2^{n+1}} \mathbf{R}_S\left(\tilde{p}_1^n, \tilde{p}_2^n\right) dt + \int_{t_2^{n+1}}^{t_1^{n+1}} \mathbf{R}_S\left(\tilde{p}_1^n, \tilde{p}_2^{n+1}\right) dt.$$

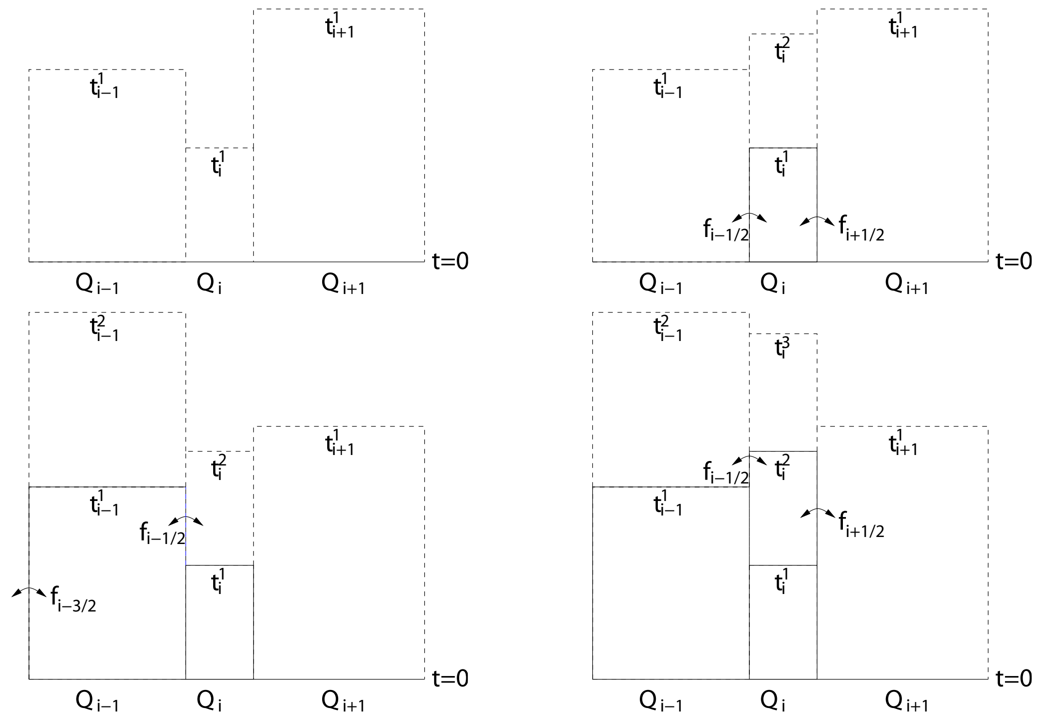


Figure 4.5: Sequence of steps 1-4 of a computation with 3 different elements and local time-stepping

Chapter 5

Solving equation systems

The application of an implicit scheme for the Navier-Stokes equations leads to a nonlinear or, in the case of Rosenbrock methods, linear system of equations. To solve systems of this form, differents methods are known and are used. As mentioned in the introduction, the question of efficiency of an implicit scheme will be decided by the solver for the equation systems. Therefore, this chapter is the longest of this book.

The outline is as follows: We will first describe properties of the systems at hand and general paradigms for iterative solvers. Then we will discuss methods for nonlinear systems, namely fixed point methods, multigrid methods and different variants of Newton's method. In particular, we make the point that dual time stepping multigrid as currently applied in industry can be vastly improved and that inexact Newton methods are an important alternative. We will then discuss Krylov subspace methods for the solution of linear systems and finally revisit Newton methods in the form of the easy to implement Jacobian-free Newton-Krylov methods.

5.1 The nonlinear systems

For DIRK or BDF methods we obtain systems of the form (see (4.21), (4.23) and (4.12))

$$\underline{\mathbf{u}} = \underline{\tilde{\mathbf{u}}} + \alpha \Delta t \hat{\underline{\mathbf{f}}}(\underline{\mathbf{u}}), \tag{5.1}$$

where $\underline{\mathbf{u}} \in \mathbb{R}^m$ is the vector of unknowns, α a parameter and $\underline{\tilde{\mathbf{u}}}$ is a given vector. As before, the underbar denotes a vector of all conservative variables from all cells. Finally, the function $\hat{\underline{\mathbf{f}}}(\underline{\mathbf{u}})$ consists of everything else coming from the spatial and temporal discretization. In the case of an autonomous ODE, $\hat{\underline{\mathbf{f}}}(\underline{\mathbf{u}})$ just denotes an evaluation of the function $\underline{\mathbf{f}}(\underline{\mathbf{u}})$ representing the spatial discretization on the whole grid.

If (5.1) has a unique solution, we call it $\underline{\mathbf{u}}^*$. For nonlinear systems, there are two important formulations that are used depending on the context. The first one is the fixed point form

$$\underline{\mathbf{u}} = \underline{\mathbf{g}}(\underline{\mathbf{u}}) \tag{5.2}$$

and the second one is the root form

$$\underline{\mathbf{F}}(\underline{\mathbf{u}}) = \underline{\mathbf{0}}. \tag{5.3}$$

Equation (5.1) is for example in fixed point form and one possible root form would be

$$\underline{\mathbf{u}} - \underline{\tilde{\mathbf{u}}} - \alpha\Delta t\hat{\underline{\mathbf{f}}}(\underline{\mathbf{u}}) = \underline{\mathbf{0}}.$$

Obviously, neither form is unique for a given nonlinear equation.

For the right hand side functions arising in CFD, formulas for exact solutions of (5.1) do not exist, which is why iterative methods are needed. These produce a sequence of iterates $\{\underline{\mathbf{u}}^{(k)}\}$, hopefully converging to the solution $\underline{\mathbf{u}}^*$. There exists a plethora of schemes to solve multidimensional nonlinear systems, for example fixed point methods, Newton's method and its variants or multigrid methods. All of these are employed in the CFD context, with fixed point methods being used due to high robustness, meaning global and provable convergence, whereas the other two are much faster. We will examine those in detail in this chapter. First, we will discuss some properties of the nonlinear equation system.

Important mathematical questions are if there exist solutions of a given equation and if these are unique. As mentioned in chapter 2, both of these have not been answered for the continuous equations. In fact, it is known that the steady Euler equations allow nonunique solutions and for the Navier-Stokes equations, existence and uniqueness results for general data are an open question. With regard to the discrete equations we are faced with, we know that if the right hand side is Lipschitz continuous, then the later presented Banach's fixed point theorem tells us that for a sufficiently small Δt, equation (5.1) has a unique solution. For larger time step sizes or right hand sides that are not Lipschitz continuous, no results are known.

Regarding the properties of the discrete system, an important role is played by numerical flux functions, of which we required that they are consistent, which in particular implied Lipschitz continuity, but not differentiability! However, a look at typical numerical flux functions tells us that they are differentiable except for a finite number of points. In the finite volume context, we have to look at the reconstruction and the limiters as well. The first one is based on linear least squares problems, which means that the reconstructed values depend differentiably on the data. On the other hand, the latter always contain minimum or maximum functions and are therefore differentiable except at a finite number of points and Lipschitz continuous otherwise. Regarding discontinuous Galerkin methods, we know that the cellwise approximations are even differentiable and the only problem is posed by the numerical flux functions. Finally, we have to consider turbulence models and these are again in general only Lipschitz continuous and piecewise differentiable.

All these components are then combined using sums and products, meaning that the resulting function is globally Lipschitz continuous and except for a finite number of points, also differentiable.

5.2 The linear systems

In the case of a Rosenbrock method (4.30) or if the nonlinear systems are solved using Newton's method, we obtain linear systems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.4}$$

with $\mathbf{x}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{A} \in \mathbb{R}^{m\times m}$ with

$$\mathbf{A} = \left.\frac{\partial \underline{\mathbf{F}}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}\right|_{\underline{\mathbf{u}}} = \left.\left(\mathbf{I} - \alpha\Delta t \frac{\partial \hat{\underline{\mathbf{f}}}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}\right)\right|_{\underline{\mathbf{u}}}. \tag{5.5}$$

The values of the Jacobian and thus the cost of computing it depend on the set of unknowns chosen, e.g. primitive or conservative. Often, the formulas are simpler when using primitive variables, leading to a small but noticable speedup.

The matrix $\mathbf{A}$ in (5.5) has the property of being sparse, ill conditioned and unsymmetric (though it has a certain block symmetry). We can also deduce that the matrix is close to a block diagonal matrix for small time steps and that thus the linear equation systems become the harder to solve, the bigger the time step is.

As for the block structure of $\mathbf{A}$, this depends on the space discretization method. The size of the blocks corresponds to the number of unknowns in a cell. For a finite volume scheme, the size is $d + 2$ and the blocks are dense, whereas for a modal DG scheme, the size increases to $(p + d)!/(p!d!)$ with again dense blocks. However, while the block size for the DG-SEM is even greater for the same order with $(p + 1)^d$ unknowns, the blocks are suddenly sparse with $(d + 1)dp^d$ nonzero entries for the case of the Euler equations. This can be immediately seen from (3.35) and (3.36) and is visualized in figure 5.1.

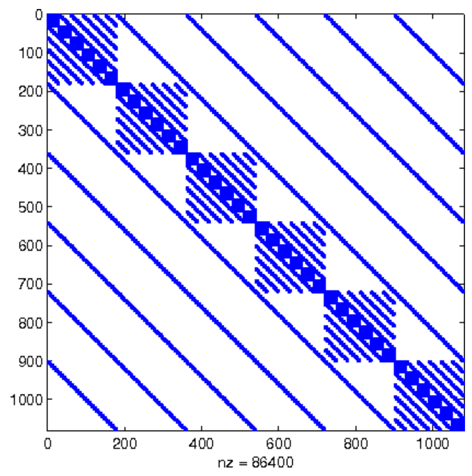


Figure 5.1: Sparsity pattern of a diagonal block for the Euler equations.

In the case of the Navier-Stokes equations, the computations of the gradients leads to a less sparse block. However, due to using the dGRP-flux (3.43), we still have some sparsity, as shown in figure 5.2. Regarding the offdiagonal blocks, we again have to take into account the gradients of the points on the boundary in neighboring cells.

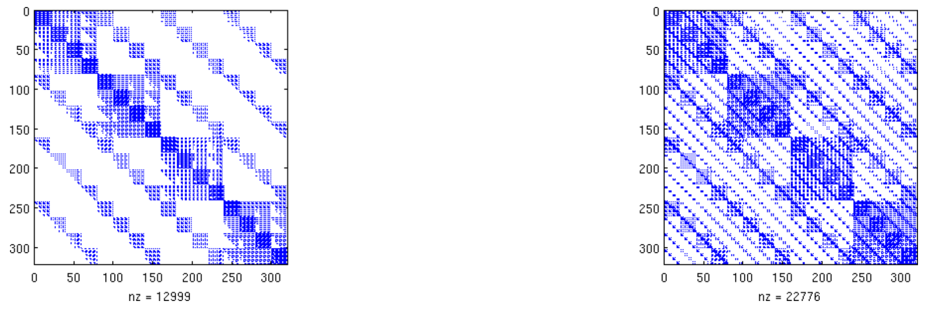


Figure 5.2: Sparsity pattern of a diagonal (left) and offdiagonal (right) for the Navier-Stokes equations.

Since the number of unknowns is rather large and can be several millions in the case of 3D-Navier-Stokes computations, it is of utmost importance to use the structure of this equation system to obtain a viable method. Thereby, both storage and computing time need to be considered. For example, it is impossible to store a dense Jacobian matrix and therefore, the sparsity of the Jacobian must be exploited. This narrows the choice down to sparse direct methods, splitting methods, Krylov subspace methods and linear multigrid methods.

Splitting methods like Jacobi or Gauss-Seidel can exploit the sparsity, but they are only linearly convergent with a constant near one, which makes them too slow. However, they will play a role as preconditioners and smoothers for other methods. As for linear multigrid methods, there is no theory that tells us what smooth components are for the Euler- or Navier-Stokes equations and therefore, no fast linear multigrid solver has been found. Regarding sparse direct methods, there has been significant progress during the last decades and robust solver packages like PARDISO, SuperLU or UMFPACK have been developed. At the moment, these are still slower than Krylov subspace methods for large systems, but might be an option in the future [87]. The major choice left are therefore so called Krylov subspace methods.

5.3 Rate of convergence and error

Since for both nonlinear and linear systems, iterative methods play an important role, it helps to have properties that can be used to compare these schemes. One is the notion of the rate of convergence.

Definition 7 (Rate of convergence) *A method with iterates* $\mathbf{x}^{(k)} \in \mathbb{R}^m$, $k \in \mathbb{N}$, *which converges to* $\mathbf{x}^*$ *is called*

- *linearly convergent to* $\mathbf{x}^*$, *if* $\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq C\|\mathbf{x}^{(k)} - \mathbf{x}^*\|$, $0 < C < 1$,
- *superlinearly convergent of order* p *to* $\mathbf{x}^*$, *if* $\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq C\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^p$ *with* $p > 1$, $C > 0$,
- *superlinearly convergent to* $\mathbf{x}^*$, *if* $\lim_{k\to\infty} \frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\|}{\|\mathbf{x}^{(k)} - \mathbf{x}^*\|} = 0$,
- *quadratically convergent to* $\mathbf{x}^*$, *if* $\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq C\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^2$ *with* $C > 0$.

Note that superlinear convergence of order p implies superlinear convergence, but not the other way around.

Using the error for a given vector $\underline{\mathbf{u}}$

$$\underline{\mathbf{e}} := \underline{\mathbf{u}} - \underline{\mathbf{u}}^* \tag{5.6}$$

and using the short hand notation for the error in the k-th iteration

$$\underline{\mathbf{e}}_k := \underline{\mathbf{u}}^{(k)} - \underline{\mathbf{u}}^*, \tag{5.7}$$

we can write these notions in quite compact form.

5.4 Termination criterias

For iterative schemes, it is important to have a termination criterion that avoids costly oversolving, meaning beyond the accuracy needed. Furthermore, a norm has to be chosen. Often the 1-, 2- or ∞-norm are used with a fixed tolerance for each system, to be defined in the parameter file. While easy, this neglects that we do not solve these nonlinear system for themselves, but inside a time adaptive implicit time integration. This means that the nonlinear systems have to be solved to a degree that guarantees not to interfere with the error estimator in the time integration, such as not to lead to unpredictable behavior there. In [180], it is demonstrated on the DASSL code, how a bad choice of the tolerance in the nonlinear solver can lead to problems in the adaptive time integrator. The way to circumvent this is to choose the tolerances in the nonlinear solver based on the tolerances in the time integrator (see (4.35)) and furthermore, to measure convergence in the same norm as for the error estimator. Thus, we will always assume that a tolerance TOL and a norm $\|\cdot\|$ will be given from the outside solver.

We are now faced with the same problem as for the time integration: We do not know the error and have to estimate it instead. In the absence of any other information about the system, typically one of the following two quantities is used to estimate the error: The residual $\|\underline{\mathbf{F}}(\underline{\mathbf{u}}^{(k)})\|$ given by (5.3) or the difference of two subsequent iterates $\|\underline{\mathbf{u}}^{(k+1)} - \underline{\mathbf{u}}^{(k)}\|$. Obviously, both converge to zero if the sequence converges to $\underline{\mathbf{u}}^*$, but conversely, it is not true that when one of these is small, we can be sure that the sequence converges to the solution or just that $\underline{\mathbf{u}}^{(k)}$ is close to $\underline{\mathbf{u}}^*$. Nevertheless, the term convergence criteria is often used instead of termination criteria and it is common to say that a solver has converged to denote that the termination criterion is satisfied.

However, for a differentiable $\underline{\mathbf{F}}$ we have the following relation between relative residual and relative error:

$$\frac{\|\mathbf{e}\|}{4\|\mathbf{e}_0\|\kappa(\underline{\mathbf{F}}'(\underline{\mathbf{u}}^*))} \leq \frac{\|\underline{\mathbf{F}}(\underline{\mathbf{u}})\|}{\|\underline{\mathbf{F}}(\underline{\mathbf{u}}^{(0)}\|} \leq \frac{4\kappa(\underline{\mathbf{F}}'(\underline{\mathbf{u}}^*))\|\mathbf{e}\|}{\|\mathbf{e}_0\|},$$

where κ denotes the condition number in the 2-norm. The inequalities mean that if the Jacobian of the function in the solution is well conditioned, the residual is a good estimator for the error. If this is not the case, then we do not know. For the other criteria, we have that

$$\mathbf{e}_{k+1} = \mathbf{e}_k + \underline{\mathbf{u}}^{(k+1)} - \underline{\mathbf{u}}^{(k)}$$

and thus for a method that is convergent of order p we have

$$\|\mathbf{e}_k\| = \|\underline{\mathbf{u}}^{(k+1)} - \underline{\mathbf{u}}^{(k)}\| + \mathcal{O}(\|\mathbf{e}_k\|^p)$$

near the solution.

As termination criteria, we always use relative ones. For the residual based indicator we obtain

$$\|\underline{\mathbf{F}}(\underline{\mathbf{u}}^{k+1})\| \leq TOL \cdot \underline{\mathbf{F}}\|(\underline{\mathbf{u}}^{(0)})\| \tag{5.8}$$

and for the solution based indicator the test becomes

$$\|\underline{\mathbf{u}}^{(k+1)} - \underline{\mathbf{u}}^{(k)}\| \leq TOL\|\underline{\mathbf{u}}^{(0)}\|. \tag{5.9}$$

For the solution of steady problems when using a time integration scheme to compute the steady state, it is typically sufficient to do only a very small number of iterations of the nonlinear solver. This is not the case for unsteady problems, where it is important that the termination criterion is reached for not interfering with the outer time integration scheme.

Now, it may happen that the timestep is chosen so large by the adaptive time step selector, that the nonlinear solver does not converge. Therefore, it is useful to add another feedback loop to this: Set a maximal number of iterations for the nonlinear solver. If the iteration has not passed the termination criterion by then, repeat the time step, but divide the step size by two.

5.5 Fixed Point methods

As mentioned before, we speak of a fixed point equation, if it is in the form

$$\underline{\mathbf{g}}(\underline{\mathbf{u}}) = \underline{\mathbf{u}}. \tag{5.10}$$

Note that (5.1) is in fixed point form. A method to solve such equations is the fixed point iteration

$$\underline{\mathbf{u}}^{(k+1)} = \underline{\mathbf{g}}(\underline{\mathbf{u}}^{(k)}). \tag{5.11}$$

A very useful and important theorem to determine convergence of this type of methods is Banach's fixed point theorem:

Theorem 4 *Let X be a Banach space with norm $\|\cdot\|$. Let $D \subset X$ be a closed set and $\underline{\mathbf{g}}$ be a contraction on D, meaning that $\underline{\mathbf{g}}$ is Lipschitz continuous on D with Lipschitz constant $L_g < 1$. Then the fixed point equation (5.10) has a unique solution $\underline{\mathbf{u}}^*$ and the iteration (5.11) converges linearly to this solution if $\underline{\mathbf{u}}^{(0)} \in D$.*

Since Δt appears in the definition of equation (5.1), a bound of the form $\mathcal{O}(\Delta t L_g) < 1$ has to be put on the time step to guarantee convergence. If the problem is very stiff, this leads to an unacceptably strict constraint, even for A-stable methods.

5.5.1 Splitting Methods

A particular type of fixed point methods for the solution of linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ (5.4) are splitting methods. The name comes from writing the system matrix in a split form

$$\mathbf{A} = (\mathbf{A} - \mathbf{B}) + \mathbf{B}$$

via a matrix $\mathbf{B} \in \mathbb{R}^{m\times m}$ and then defining a fixed point equation, which is equivalent to $\mathbf{A}\mathbf{x} = \mathbf{b}$ via

$$\mathbf{x} = (\mathbf{I} - \mathbf{B}^{-1}\mathbf{A})\mathbf{x} + \mathbf{B}^{-1}\mathbf{b}.$$

The corresponding fixed point method is given by

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{B}^{-1}\mathbf{A})\mathbf{x}^{(k)} + \mathbf{B}^{-1}\mathbf{b}. \tag{5.12}$$

Several specific choices for the matrix $\mathbf{B}$ correspond to well known methods. The idea of the Gauß-Seidel method is to start with the first equation and solve for the first unknown, given the current iterate. The first component of the first iterate is then overwritten and the method proceeds to the next equation, until each equation has been visited. Thus, the method is inherently sequential and furthermore, if the system matrix is triangular, it is a direct solver. In matrix notation, it can be written as

$$\mathbf{x}^{(x+1)} = -(\mathbf{L} + \mathbf{D})^{-1}\mathbf{U}\mathbf{x}^{(k)} + (\mathbf{L} + \mathbf{D})^{-1}\mathbf{b}, \tag{5.13}$$

where $\mathbf{L}$ is the strict lower left part of $\mathbf{A}$, $\mathbf{D}$ the diagonal and $\mathbf{U}$ the strict upper right part. Thus, $\mathbf{B} = \mathbf{L} + \mathbf{D}$.

If, following the forward sweep of the Gauß-Seidel method, a backward sweep using the same method with a reverse numbering of the unkowns is performed, the symmetric Gauß-Seidel method (SGS) is obtained. This can be written using

$$\mathbf{B}_{SGS} = (\mathbf{D}+\mathbf{U})^{-1}\mathbf{D}(\mathbf{D}+\mathbf{L})^{-1}. \tag{5.14}$$

Note that the convergence behavior of both GS and SGS depends on the ordering of the unknowns. Therefore, if this method is used in some way, we suggest to reorder the unknowns appropriately after grid generation or adaption. A particular strategy is to use physical reordering as suggested in [143]. There, planes are built orthogonal to the direction of the inflow and then the unknowns in the first plane are numbered first, then the second plane, and so on. This significantly improves the performance of SGS, as well as of the later described ILU preconditioner.

Furthermore, there is the Jacobi method, which is inherently parallel and is obtained by chosing

$$\mathbf{B}_J = \mathbf{D}. \tag{5.15}$$

The idea behind the method is to solve all equations simultaneously for the unknown on the diagonal, given the current values for the other unknowns. In matrix notation, it can be written as

$$\mathbf{x}^{(x+1)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}. \tag{5.16}$$

Finally, all these methods exist in a relaxation form, meaning that a relaxation parameter ω is added, which can lead to improved convergence speed. For SGS, this method is known as the symmetric overrelaxation method (SSOR) and is given by

$$\mathbf{B}_{SSOR} = \frac{1}{\omega(2-\omega)}\mathbf{B}_{SGS} = \frac{1}{\omega(2-\omega)}(\mathbf{D}+\mathbf{U})^{-1}\mathbf{D}(\mathbf{D}+\mathbf{L})^{-1}. \tag{5.17}$$

All these methods can be extended trivially to block matrices.

Convergence of these methods can be analyzed using the following corollary of Banach's fixed point theorem:

Corollary 5 *Let $\mathbf{x}^{(k+1)} = \mathbf{M}\mathbf{x}^{(k)} + \mathbf{z}$ be a linear iterative scheme. Then the scheme is globally convergent if and only if $\rho(\mathbf{M}) < 1$.*

This can be proved using that for any $\epsilon > 0$, there is a matrix norm, such that for any matrix $\mathbf{M} \in \mathbb{R}^{m\times m}$, it holds that $\|\mathbf{M}\| \leq \rho(\mathbf{M}) + \epsilon$.

Thus, if $\|\mathbf{I}-\mathbf{B}^{-1}\mathbf{A}\| < 1$ or more general if $\rho(\|\mathbf{I}-\mathbf{B}^{-1}\mathbf{A}\|) < 1$, the methods converge. In the case of block matrices, this was considered by Varga [211] and in the context of computational fluid dynamics, by Dwight [54]. Furthermore, we have that the methods actually converge to the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$, due to the special form of the vector $\mathbf{z}$ in these cases.

5.5.2 Fixed point methods for nonlinear equations

The Jacobi and Gauß-Seidel iteration can be extended to nonlinear equations [165]. To this end, we recall the original ideas of the method. Thus, the Gauß-Seidel process starts with the first nonlinear equation and, given the current iterate $\underline{\mathbf{u}}^{(k)}$, solves for the first unknown $u_1^{(k)}$. The first component of the first iterate is then overwritten and the method proceeds to the next equation, until each equation has been visited. Here, solve means an inner iterative process for the solution of nonlinear systems, for example Newton's method. Sometimes, the inner solve is added to the name, for example, Gauß-Seidel-Newton process. We thus obtain the method

- For $k = 1, \dots$ until convergence do
  - For $i = 1, \dots, m$ do:

$$\text{Solve } f_i(u_1^{(k+1)}, \dots, u_{i-1}^{(k+1)}, u_i^{(k+1)}, u_{i+1}^{(k)}, \dots u_n^{(k)}) = 0 \text{ for } u_i^{(k+1)}. \tag{5.18}$$

The symmetric Gauß-Seidel and the Jacobi method can be extended in a corresponding way, for example the Jacobi process can be written as

- For $k = 1, \dots$ until convergence do
  - For $i = 1, \dots, n$ do:

$$\text{Solve } f_i(u_1^{(k)}, \dots, u_{i-1}^{(k)}, u_i^{(k+1)}, u_{i+1}^{(k)}, \dots u_n^{(k)}) = 0 \text{ for } u_i^{(k+1)}. \tag{5.19}$$

Again, for all of these processes, relaxation can be used to improve convergence. Furthermore, the extension to a block form, where instead of scalar equations, nonlinear systems have to be solved in each step, is straight forward.

Regarding convergence, we can again use Banach's fixed point theorem to deduce under the assumption of exact solves, that this will be linear. More precise, we define a mapping $\mathbf{g}$ for the Gauß-Seidel case via the formula

$$g_i(\mathbf{x}, \mathbf{y}) := f_i(x_1, ..., x_i, y_{i+1}, ..., y_n)$$

for it's i-th component. Then we are in a fixed point of the original equation if $\mathbf{x} = \mathbf{y}$, corresponding to $\underline{\mathbf{u}}^{(k+1)} = \underline{\mathbf{u}}^{(k)}$. It then follows using the implicit function theorem that the process converges in a neighborhood of the solution $\underline{\mathbf{u}}^*$ if

$$\rho(-\partial_1 \mathbf{g}(\underline{\mathbf{u}}^*, \underline{\mathbf{u}}^*)^{-1} \partial_2 \mathbf{g}(\underline{\mathbf{u}}^*, \underline{\mathbf{u}}^*)) < 1,$$

where the partial derivatives are taken with respect to $\mathbf{x}$ and $\mathbf{y}$ [165, p. 31]. This corresponds to the condition on $\mathbf{I} - \mathbf{B}^{-1}\mathbf{A}$ in the linear case.

5.6 Multigrid methods

A class of methods that has been developed particularly for equation systems arising from discretized partial differential equations are multigrid methods [80, 201, 222]. If designed properly, multigrid schemes are linearly convergent and furthermore, the so called textbook multigrid efficiency has been demonstrated or even proved for large classes of partial differential equations, in particular elliptic ones. This means that the convergence rate is independent of the mesh width and that only a few steps are necessary to compute the solution. Multigrid methods are the standard method used in industry codes, even though textbook multigrid efficiency has not been acchieved for the Navier-Stokes equations.

The idea is to divide the error of the current iterate into two parts, called smooth and nonsmooth or sometimes low and high frequency. The latter part is taken care of by a so called smoother $\mathbf{S}$ and the other part by the coarse grid correction, which solves the suitably transformed problem in a space with fewer unknowns using the same approach again, thus leading to a recursive method on multiple grids. The point here is to choose the coarse space such that the smooth error can be represented well in that space and thus the dimension of the problem has been significantly decreased.

There are basically three concepts to choose the coarse space. The first is the original one, where the computational grid is coarsened in some way. Then there are algebraic multigrid methods, which just use algebraic operations to reduce the number of unknowns. These are also generalized in the numerical linear algebra community to multilevel methods. Finally, in particular for DG methods, there are multi-p methods (also called p-multigrid), that use lower order discretizations to reduce the space dimension.

We will now first explain the multigrid method for linear problems, before looking at nonlinear problems and then multi-p methods. After that, we will demonstrate the benefit of designing a multigrid scheme directly for unsteady flows on a model equation and Runge-Kutta smoothers.

To describe the method, assume that a hierarchy of spaces is given, denoted by their level l, where a smaller index corresponds to a smaller space. Corresponding to this are a restriction operator $\mathbf{R}_{l-1,l}$ to go from level l to the next coarse level $l-1$ and a prolongation operator $\mathbf{P}_{l,l-1}$ for the return operation.

5.6.1 Multigrid for linear problems

The multigrid method was first designed for the Poisson equation. Consider the Dirichlet problem

$$\begin{aligned} -u_{xx} &= f(x), \quad x \in (0,1),\, f \in \mathcal{C}((0,1),\mathbb{R}) \\ u(x) &= 0,\, x = 0, x = 1. \end{aligned}$$

The eigenfunctions of $-u_{xx}$ satisfying the boundary conditions are

$$\phi(x) = \sin(k\pi x), k \in \mathbb{N},$$

which are sine-functions of increasing oscillation. If the problem is discretized with m unknowns using second order central differences and a fixed mesh width Δx, then the resulting matrix has eigenvectors that are discrete evaluations of the eigenfunctions for the first m frequencies. Now the point is that if we define a coarse grid by dropping every other point, then only the slowly varying, namely the more smooth functions can be represented there. Furthermore, it can be shown that the spectral radius of the iteration matrices for Jacobi and Gauß-Seidel approaches one as the mesh is refined, but that they are very good at reducing the highly oscillatory error components. It is from this observation that the vocabulary with smooth and nonsmooth errors was derived.

If a different linear PDE is considered, the design of a multigrid method for that equation repeats the steps from above: First determine the eigenfunctions of the continuous operator and their discrete counterparts for the specific discretization given. Then define the coarse space and determine the eigenfunctions/eigenvectors present in the coarse space. With that information, find a smoother that takes care of the other components of the error.

This means that the notions of high and low frequency errors depend on the equation to be solved and the discretization used. Correspondingly, the components of the method (smoother, restriction and prolongation) have to be chosen in an appropriate way and the same smoother may not work for a different equation. In the case that we do not consider a linear, but a nonlinear PDE and the linear equation system comes from a linearization, e.g. from Rosenbrock time integration or Newton's method, the design of multigrid methods in this way becomes even more difficult.

For conservation and balance laws, it is important to have restriction and prolongation operators that are conservative. To this end, a coarse grid is obtained by agglomerating a number of neighboring cells, giving rise to the term agglomeration multigrid. The restricted value in a coarse cell is then given by summing up the fine grid values, weighted by the volumes of the respective cells and dividing by the total volume. For an equidistant grid in one dimension, the corresponding restriction operator would be given by

$$\mathbf{R}_{l-1,l} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 0 & \dots & 0 & \\ 0 & 0 & 1 & 1 & 0\dots & 0 \\ \vdots & \vdots & & & & \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

One possible prolongation is the injection, where the value in the coarse cell is taken as value on all the corresponding fine cells, which would result in $\mathbf{P}_{l,l-1} = 2\mathbf{R}^T_{l-1,l}$. This can be refined by interpolating means of reconstructed functions. Jameson suggests a bilinear interpolation, which in addition to the coarse grid point corresponding to a fine grid point incorporates those neighboring coarse grid cells which would incorporate a fine grid neighbor of the fine grid point considered, using suitable weights.

Furthermore, the problem on the coarse grids has to be defined. A common way for linear problems is to use

$$\mathbf{A}_{l-1} = \mathbf{R}_{l-1,l}\mathbf{A}_l\mathbf{P}_{l,l-1}.$$

Alternatively, the problem can be discretized directly on the coarse level, which will be the way to go for nonlinear problems. To obtain the problem on the coarse level, we now use that for linear problems if $\mathbf{Ax} - \mathbf{b} = \mathbf{r}$, then

$$\mathbf{Ae} = \mathbf{r}. \tag{5.20}$$

This is called the defect equation and means that on the coarse level, we will solve for the error and then correct the fine level solution by the prolongated error on the coarse level.

Finally, we the obtain the following scheme to solve the system $\mathbf{Ax} = \mathbf{b}$:

Function MG($\mathbf{x}_l, \mathbf{b}_l, l$)

- if ($l = 0$), $\mathbf{x}_l = \mathbf{A}_l^{-1}\mathbf{b}_l$ (Exact solve on coarse grid)
- else
  - $\mathbf{x}_l = \mathbf{S}_l^{\nu_1}(\mathbf{x}_l, \mathbf{b}_l)$ (Presmoothing)
  - $\mathbf{r}_{l-1} = \mathbf{R}_{l-1,l}(\mathbf{b}_l - \mathbf{A}_l\mathbf{x}_l)$ (Restriction)
  - $\mathbf{v}_{l-1} = 0$
  - For ($j = 0$; $j < \gamma$; $j++$) $MG(\mathbf{v}_{l-1}, \mathbf{r}_{l-1}, l-1)$ (Computation of the coarse grid correction)
  - $\mathbf{x}_l = \mathbf{x}_l + \mathbf{P}_{l,l-1}\mathbf{v}_{l-1}$ (Correction via Prolongation)
  - $\mathbf{x}_l = \mathbf{S}_l^{\nu_2}(\mathbf{x}_l, \mathbf{b}_l)$ (Postsmoothing)
- end if

If $\gamma = 1$, we obtain a so called V-cycle, where each grid is visited only once, whereas if $\gamma = 2$, we obtain a W-cycle. Both are illustrated in figure 5.6.1. Larger choices of γ will not be used, since there is rarely a benefit for the additional cost.

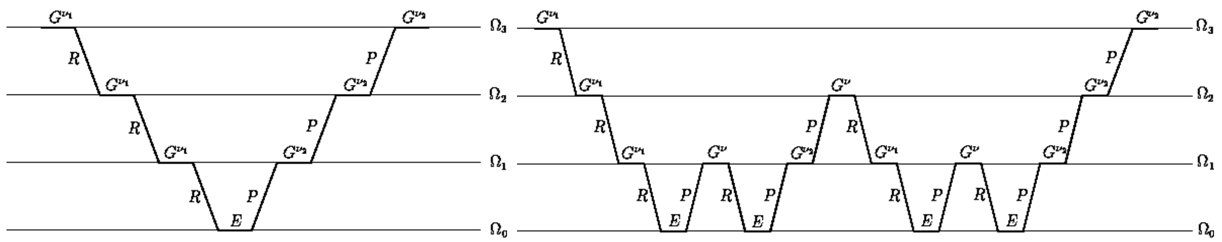


Figure 5.3: V and W-multigrid cycle

Regarding convergence theory, note that the iteration matrix of the scheme for two grids only can be written in compact form as

$$\mathbf{M} = \mathbf{S}_l^{\nu_2}(\mathbf{I} - \mathbf{P}_{l,l-1}\mathbf{A}_l^{-1}\mathbf{R}_{l-1,l}\mathbf{A}_l)\mathbf{S}_l^{\nu_1} \tag{5.21}$$

and thus, corollary 5 says that the method converges if $\rho(\mathbf{M}) < 1$. The common approach to prove this and in particular that $\rho(\mathbf{M})$ is independent of the mesh width, is the analysis as established by Hackbusch [80], which distinguishes between the smoothing and the approximation property. The first one describes if the smoothing operator is able to deal with the high frequency errors and the approximation property describes if the coarse grid solution is able to approximate the remaining components of the error.

5.6.2 Full Approximation Schemes

If instead of a linear problem, a nonlinear equation $\mathbf{f}(\underline{\mathbf{u}}) = \mathbf{s}$ is considered, the multigrid method has to be modified and is called Full Approximation Scheme (FAS). This type of method is widespread in industry, in particular for steady flows, where text book multigrid efficiency has been demonstrated for the steady Euler equations [34] and at least mesh width indepent convergence rates for the Navier-Stokes equations. The point where we have to modify the scheme for the linear case is that for a nonlinear operator, we have in general

$$\mathbf{f}(\underline{\mathbf{u}}) - \mathbf{f}(\underline{\mathbf{u}}^*) \neq \mathbf{f}(\mathbf{e})$$

and therefore the defect equation is not valid and we cannot solve for the coarse grid error directly. Therefore, we instead consider the equation

$$\mathbf{f}(\underline{\mathbf{u}}^*) = \mathbf{f}(\underline{\mathbf{u}}) - \mathbf{r}(\underline{\mathbf{u}}), \tag{5.22}$$

which reduces to the coarse grid equation from the last section for the linear case. Here, care has to be taken that due to the nonlinearity, the right hand side might be such that it is not in the image of $\mathbf{f}$. In this case, the residual needs to be damped by a factor s. However, this does not seem to be necessary in our case. There is one more change, because even on the coarsest level, it is impossible to solve the nonlinear equation exactly. Therefore, only a smoothing step is performed there. The prolongation $\mathbf{Q}_{l,l+1}$ and restriction $\mathbf{R}_{l+1,l}$ are chosen as described in the last section and the coarse grid problem is defined in a natural way as if the equation would be discretized on that grid. We thus obtain the following method, where $\underline{\tilde{\mathbf{u}}}_l$ is the current approximation and $\underline{\mathbf{u}}_l$ the output:

Function FAS-MG($\underline{\tilde{\mathbf{u}}}_l, \underline{\mathbf{u}}_l, \mathbf{s}_l, l$)

- $\underline{\mathbf{u}}_l = \mathbf{S}_l^{\nu_1}(\underline{\tilde{\mathbf{u}}}_l, \mathbf{s}_l)$ (Presmoothing)
- if ($l > 0$)
  - $\mathbf{r}_l = \mathbf{s}_l - \mathbf{f}_l(\underline{\mathbf{u}}_l)$
  - $\tilde{\mathbf{u}}_{l-1} = \mathbf{R}_{l-1,l}\underline{\mathbf{u}}_l$ (Restriction of solution)
  - $\mathbf{s}_{l-1} = \mathbf{f}_{l-1}(\tilde{\mathbf{u}}_{l-1}) + \mathbf{R}_{l-1,l}\mathbf{r}_l$ (Restriction of residual)

- For $(j = 0;\ j < \gamma;\ j++)$ FAS-MG$(\tilde{\mathbf{u}}_{l-1}, \underline{\mathbf{u}}_{l-1}, \mathbf{s}_{l-1}, l-1)$ (Computation of the coarse grid correction)
- $\underline{\mathbf{u}}_l = \underline{\mathbf{u}}_l + \mathbf{P}_{l,l-1}(\underline{\mathbf{u}}_{l-1} - \tilde{\mathbf{u}}_{l-1})$ (Correction via Prolongation)
- $\underline{\mathbf{u}}_l = \mathbf{S}_l^{\nu_2}(\underline{\mathbf{u}}_l, \mathbf{s}_l)$ (Postsmoothing)

• end if

Regarding smoothers, the proper choice depends on the problem. We will therefore now review the method for steady state problems, before continuing with the unsteady case.

5.6.3 Steady state solvers

In the context of compressible flows, the most widely used and fastest FAS type scheme is the one developed by Jameson over the course of thirty years [99]. The latest version is due to Jameson and Caughey [34] and solves the steady Euler equations around an airfoil in three to five steps. Thus, two dimensional flows around airfoils can be solved on a PC in a matter of seconds. The solution of the steady RANS equations is more difficult and takes between fifty steps and two hundred steps for engineering accuracies.

As smoothers, different methods have been successfully used, namely explicit RK schemes, explicit additive RK methods, point implicit smoothers, line implicit smoothers and the SGS method.

For the Runge-Kutta methods, recall that a method to compute a steady state is to advance an approximation of the steady state in time using a time stepping method. Thus, an explicit Runge-Kutta method is an iterative method for the solution of steady state problems and can be used as a smoother. The scheme is applied to the steady state equation, with an added time derivative and the forcing term $\mathbf{s}_l$ from the FAS-MG:

$$\frac{d}{dt}\underline{\mathbf{u}}_l = \mathbf{f}_l(\underline{\mathbf{u}}_l) - s_l.$$

Calling the initial value $\underline{\mathbf{u}}_l^{(0)}$ and using a low storage explicit RK method (4.18), we obtain the scheme

$$\begin{aligned}
\underline{\mathbf{u}}_l^{(1)} &= \underline{\mathbf{u}}_l^{(0)} - \alpha_1 \Delta t_l [\mathbf{f}_l(\underline{\mathbf{u}}^{(0)}) + \mathbf{s}_l] \\
&\dots \\
\underline{\mathbf{u}}_l^{(q+1)} &= \underline{\mathbf{u}}_l^{(0)} - \alpha_{q+1} \Delta t_l [\mathbf{f}_l(\underline{\mathbf{u}}^{(q)}) + \mathbf{s}_l] \\
\underline{\mathbf{u}}_l^{n+1} &= \underline{\mathbf{u}}_l^{(q+1)}.
\end{aligned}$$

Jameson also suggests the use of additive Runge-Kutta methods (4.27), where different coefficients are used for the convective and the diffusive parts. This is done to reduce the number of evaluations of the expensive diffusive terms, but also to increase the degrees of freedom in choosing a good smoother. Given a splitting in the convective and diffusive part

$$\mathbf{f}(\underline{\mathbf{u}}) = \mathbf{f}^c(\underline{\mathbf{u}}) + \mathbf{f}^v(\underline{\mathbf{u}})$$

he suggests to implement these schemes in the following equivalent form

$$\begin{aligned}
\underline{\mathbf{u}}_l^{(0)} &= \tilde{\mathbf{u}}_l \\
\underline{\mathbf{u}}_l^{(j)} &= \tilde{\mathbf{u}}_l - \alpha_j \Delta t_l (\mathbf{f}^{c,(j-1)} + \mathbf{f}^{v,(j-1)}), \quad j = 1, ..., s \\
\underline{\mathbf{u}}_l^{n+1} &= \underline{\mathbf{u}}_l^{(s)},
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{f}^{c,(j)} &= \mathbf{f}^c(\underline{\mathbf{u}}_l^{(j)}), \quad j = 0, ..., s-1 \\
\mathbf{f}^{v,(0)} &= \mathbf{f}^v(\underline{\mathbf{u}}_l^{(0)}), \\
\mathbf{f}^{v,(j)} &= \beta_j \mathbf{f}^v(\underline{\mathbf{u}}_l^{(j)}) + (1-\beta_j)\mathbf{f}^{v,(j-1)}, \quad j = 1, ..., s-1.
\end{aligned}$$

The methods are then designed to provide an optimal damping of high frequency modes. For the one dimensional linear advection equation

$$u_t + a u_x = 0, \tag{5.23}$$

with $a > 0$, discretized using a first order upwind scheme with fixed mesh width Δx and explicit RK smoothing, this was done in [208]. However, the methods obtained in this way are not convincing when applied to the Euler or Navier-Stokes equations. Instead, Jameson used the one dimensional linear advection with a fourth order derivative to obtain a better smoother for 4th order artificial diffusion schemes [97]. The coefficients for the corresponding additive RK smoothers can be found in table 5.1 and [99].

i	1	2	3	4	5
α_i	1/3	4/15	5/9	1	-
β_i	1	1/2	0	0	-
α_i	1/4	1/6	3/8	1/2	1
β_i	1	0	0.56	0	0.44

Table 5.1: Coefficients of additive Runge-Kutta smoothers, 4-stage and 5-stage method.

In short, since the model problems are linear, the explicit Runge-Kutta method can be written as a complex polynomial $P_s(z)$ of degree s in $\Delta t\mathbf{A}$. The free parameters of the polynomial are given by the method and should be chosen, such that the value of the polynomial is minimized for the high frequency part of the spectrum of the equation. To this end, eigenvalues and eigenvectors are determined assuming periodic boundary conditions. For the linear advection equation, these are the functions $e^{ij\Theta}$, $\Theta \in [-\pi, \pi]$, where Θ is the nondimensional wave number. The high frequency components are then those that cannot be represented on the coarse grid, e.g. $|\Theta| \in [\pi/2, \pi]$. Van Leer et. al. suggest to define the smoother via the solution of the optimization problem

$$\min_{\Delta t, \alpha} \max_{|\Theta| \in [\pi/2, \pi]} |P_s(z(\Theta))|,$$

where α is the set of free parameters of the polynomial and $z(\Theta)$ are the eigenvalues of a model discretiation, namely the one dimensional linear advection equation with an upwind discretization. For this case, the optimization problem can be solved directly, but for more complex equations, this is no longer possible.

A powerful alternative to Runge-Kutta smoothers is the SGS method. This is typically used in conjunction with flux vector splitting discretizations, since this results in significantly less computational work.

Point implicit smoothers correspond to using an SDIRK scheme with only one Newton step, where the Jacobian is approximated by a block diagonal matrix [119, 120]. Line implicit smoothers are used to cope with grid induced stiffness by defining lines in anisotropic regions normal to the direction of stretching. These lines then define a set of cells, for which an SDIRK method is applied, again with one Newton step, but taking into account the coupling between the cells along the line. This leads to a tridiagonal block matrix.

Convergence is accelerated by local time stepping or preconditioning of the time derivative and furthermore residual averaging. Then, a W-cycle with four or five grid levels is performed. All of these ingredients are important for fast convergence and have been successfully tuned to the case of steady Euler flows. Residual averaging is used to increase the level of implicitness in the method and thus to move unphysical parts of the solution even faster out of the domain [100]. To this end, a small parameter ϵ is chosen and then a Laplace-filter is applied directionwise:

$$-\epsilon\bar{r}_{j-1} + (1+2\epsilon)\bar{r}_j - \epsilon\bar{r}_{j+1} = r_j. \tag{5.24}$$

This equation is solved for the new residuals $\bar{\mathbf{r}}$ using a Jacobi iteration. For a suitably large parameter ϵ, this stabilizes the scheme.

5.6.4 Multi-p methods

Another alternative in the context of DG and other higher order methods are multi-p schemes, where instead of coarsening the grid, the coarse scale problem is defined locally on each element using a lower order method [60]. Therefore, these two approaches are also often called p- and h-multigrid. Mesh width indepent convergence rates have been demonstrated numerically for steady state problems. To define the method, let a high order space $\mathbf{\Pi}_l$ and a nested low order space $\mathbf{\Pi}_{l-1} \subset \mathbf{\Pi}_l$ be given. Typically, the low order space is defined by decreasing the polynomial order by one, although in the original method, a more agressive coarsening is used, dividing the order by two. Then, the FAS multigrid method is applied as before, where we have to define new prolongations and restrictions and for the restriction of the solution (represented by the coefficients of a polynomial ansatz) and the residual, different operators $\mathbf{R}_{l-1,l}$ and $\tilde{\mathbf{R}}_{l-1,l}$ are used.

To interpolate the solution from the lower to the higher order space, we use that there is a unique representation of the basis functions ϕ_i^{l-1} of the low order space in the basis of the high order space:

$$\phi_i^{l-1} = \sum_j \alpha_{ij} \phi_j^l.$$

Then, the prolongation is given by $\mathbf{P}_{l,l-1} = (\alpha_{ij})_{ij}$. For a hierarchical basis, this representation is trivial, in that the prolongation is just an identity matrix with added zero columns.

For the restriction of the residual, $\tilde{\mathbf{R}}_{l-1,l} = \mathbf{P}_{l,l-1}^T$ is typically used. The restriction of the states is defined by an L_2-projection in that we require

$$(\mathbf{w}^{l-1}, \mathbf{v}^{l-1}) = (\mathbf{w}^{l-1}, \mathbf{v}^l), \ \forall \mathbf{w}^{l-1} \in \mathbf{\Pi}_{l-1}.$$

This leads to

$$\mathbf{R}_{l,l-1} = \mathbf{M}_{l,l-1}^{-1} \mathbf{N}_{l,l-1},$$

where $(\mathbf{M}_{l,l-1})_{ij} = (\phi_i^{l-1}, \phi_j^{l-1})$ is a quadratic matrix and $(\mathbf{N}_{l,l-1})_{ij} = (\phi_i^{l-1}, \phi_j^l)$ is rectangular. For a DG method with an orthogonal basis as for example the modal-nodal method described in section 3.8.1, we have that $(\phi_i^p, \phi_j^p) = \delta_{ij}$ and thus $\mathbf{M}_{l,l-1} = \mathbf{I}$. If a nodal basis is used, $\mathbf{M}_{l,l-1}$ is dense.

Regarding the smoother, because the coarse grid has been defined differently from before, different methods have to be chosen. Alternatives that have been used are element-Jacobi [219], line-Jacobi [60], ILU [157], Runge-Kutta methods [10, 105] and combinations thereof [129]. In the context of the linear advection equation, optimal Runge-Kutta smoothers were analyzed in [9]. There, coefficients and corresponding optimal time step sizes are computed for the case of steady flows. Again, in the unsteady case, the corresponding eigenvalues are shifted and therefore, these schemes are not optimal, let alone for more complex equations.

5.6.5 Dual Time stepping

To apply the FAS method to unsteady flows with a minimal amount of coding, Jameson developed the so called dual time stepping approach [98]. This means that a pseudo time derivative is added and the steady state of the equation system

$$\frac{\partial \mathbf{u}}{\partial t^*} + \mathbf{f}(\mathbf{u}) = \mathbf{0} \tag{5.25}$$

is computed using the multigrid method for steady states described above. Typically, $\mathbf{f}$ is the result of a BDF-time discretization, where in particular the A-stable BDF-2 method is popular. For the case of the implicit Euler as in (5.1), we obtain

$$\frac{\partial \underline{\mathbf{u}}}{\partial t^*} + \underline{\mathbf{u}}(t^*) - \underline{\mathbf{u}}^n - \Delta t \underline{\mathbf{f}}(\underline{\mathbf{u}}(t^*)) = \mathbf{0}.$$

Note that (5.25) can be solved in principle using any suitable method and *dual time stepping* is in that sense more an approach to solve a time dependent problem than a method in itself. Nevertheless, dual time stepping is inherently connected to the nonlinear multigrid method of Jameson and is thus usually meant as a nonlinear multigrid method to solve equation (5.25).

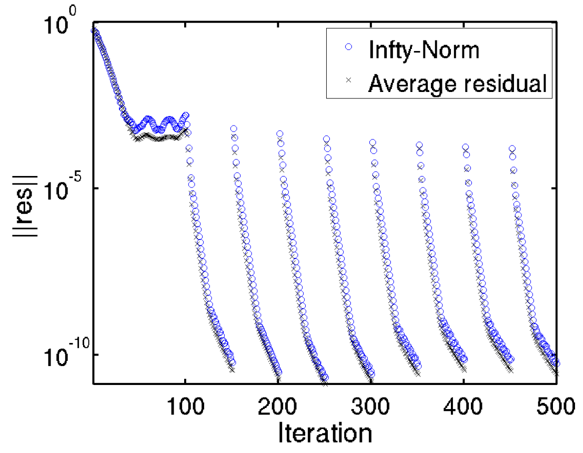


Figure 5.4: Comparison of convergence of FAS multigrid in UFLO103 for steady and unsteady flows for flow around a circle.

In figure 5.4, we demonstrate the convergence behavior of the FAS multigrid scheme implemented in the UFLO103 code of Jameson. The test case is two dimensional flow around a circle at Mach 0.25, a Reynolds number of 100,000 and zero angle of attack and the grid is a C-type grid with 512×64 unknowns. We start with freestream data and use the steady state solver for the first 100 steps. As can be seen, there is no convergence to a steady state, which is in this case due to unsteady effects starting. We then switch to the unsteady method, which uses BDF-2 and thus one nonlinear system per time step has to be solved. These systems should be easier to solve than a steady state problem and indeed, the scheme is able to reduce the norm of the residual by seven orders of magnitude in 50 steps. However, it can be seen that after about thirty iterations, the convergence rate decreases significantly.

The code of Jameson is designed for structured grids around airfoils and significantly faster than the codes employed in the industry for general unstructured grids, which are

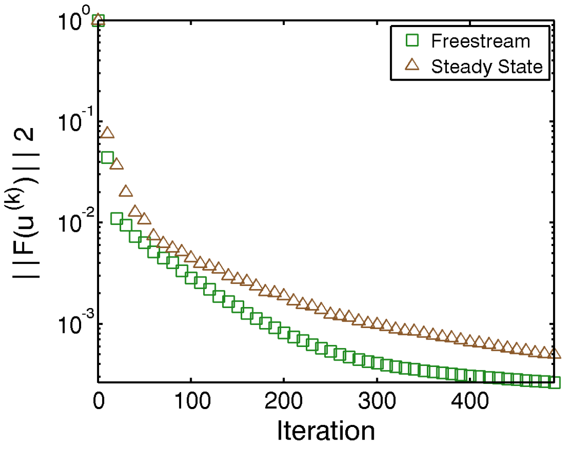


Figure 5.5: Comparison of convergence of FAS multigrid in DLR-TAU for steady and unsteady flows for flow around a circle.

designed for robustness. This is demonstrated in figure 5.5, where the convergence of the dual time stepping method implemented in the DRL TAU code is demonstrated. As can be seen, the convergence rate decreases significantly for lower tolerances. This behavior is also typical for other large industrial codes.

In practice, this leads to a scheme that takes hundreds or sometimes even thousands of iterations to solve the systems inside a time step, when we would in fact expect to be able to solve these systems much more quickly. This is because the original multigrid method is designed for steady states and the operator **f** in (5.25) is shifted due to the additional time discretization terms. Therefore, the smoother and the choice of prolongation and restriction is no longer optimized for that problem. For example, the coefficients of the steady state Runge-Kutta smoothers are optimized for a different set of eigenvalues.

5.6.6 Optimizing Runge-Kutta smoothers for unsteady flow

The lack of theory for the original method leads to a situation, where so far, there has been little success in fixing this undesirable behavior. By contrast, significant progress on the design of multigrid methods has been obtained for the incompressible Navier-Stokes equations [202].

We will consider two different ways of finding good smoothers, both involving an optimization process, which is described in [20]. This text follows that article. First of all, we can try to optimize the smoother alone, such that it removes nonsmooth error components fast. This approach was followed in [208] and later in [9]. Additionally, we suggest to compute the iteration matrix of the multigrid scheme and minimize its spectral radius in a discrete way.

The difference between the two approaches is that the latter one is theoretically able to provide truly optimal schemes, whereas the first one does not. However, the second approach is much more costly, meaning that we are not able to compute the global optimum in a reasonable time. Therefore, both approaches are discussed here.

As a model problem to design the smoothers on we consider the linear advection equation

$$u_t + au_x = 0. \tag{5.26}$$

with $a > 0$ on the interval $x \in [0, 2]$ with periodic boundary conditions. This is discretized using an equidistant finite volume method with mesh width Δx leads to the evolution equation for the cell average u_i in one cell i:

$$u_{i_t} + \frac{a}{\Delta x}(u_i - u_{i-1}) = 0.$$

Using the vector $\mathbf{u} = (u_1, ..., u_m)^T$ and

$$\mathbf{B} = \begin{pmatrix} 1 & & & & -1 \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{pmatrix},$$

we obtain the system of ODEs

$$\mathbf{u}_t + \frac{a}{\Delta x}\mathbf{B}\mathbf{u}(t) = \mathbf{0}. \tag{5.27}$$

Here, we discretize this using implicit Euler with time step size Δt, which is also a building block for the more general diagonally implicit Runge-Kutta (DIRK) methods. Thus, in each time step, a linear system has to be solved. Using the notation $\mathbf{u}^n \approx \mathbf{u}(t_n)$, this can be written as

$$\begin{aligned} \mathbf{u}^{n+1} - \mathbf{u}^n + \frac{a\Delta t}{\Delta x}\mathbf{B}\mathbf{u}^{n+1} &= 0 \\ \Leftrightarrow \mathbf{u}^n - \mathbf{A}\mathbf{u}^{n+1} &= 0 \end{aligned} \tag{5.28}$$

where

$$\mathbf{A} = \left(\mathbf{I} + \frac{\nu}{\Delta x}\mathbf{B}\right) \tag{5.29}$$

with $\nu = a\Delta t$. Here, $CFL := a\Delta t/\Delta x$ corresponds to the CFL number in the implicit Euler method. If we consider nonperiodic boundary conditions, the entry in the upper right corner of $\mathbf{B}$ becomes zero. Otherwise, additional terms appear on the right hand side, but this does not affect multigrid convergence.

If we look at the new eigenvalues and eigenvectors, we see that the eigenvectors have not been changed, but if λ_i is an eigenvalue of $\mathbf{f}$, then $1 - \nu\lambda_i$ is an eigenvalue in the unsteady case. A similar analysis has been performed in [112]. We now discuss the two optimization procedures.

5.6.7 Optimizing the smoothing properties

For the first approach, the eigenvectors of the matrix $\mathbf{A}$ from (5.29) are discrete forms of the functions $e^{ix\Theta}$ for various Θ and the eigenvalues are given by

$$\lambda(\Theta) = -1 - \frac{\nu}{\Delta x}(1 - e^{-i\Theta}).$$

If nonperiodic boundary conditions are used, the matrix becomes lower triangular and all eigenvalues are equal to $-1 - \frac{\nu}{\Delta x}$. In the steady case, the eigenvalues would be scaled and shifted, resulting in $\lambda(\Theta) = -\frac{a}{\Delta x}(1 - e^{-i\Theta})$. Now, on the coarse grid, we can represent functions with $\Theta \in [-\pi/2, \pi/2]$. Thus, the smoother has to take care of error components with $|\Theta| \in [\pi/2, \pi]$.

Due to the linearity, it is sufficient to look at $P_s(\Delta t^* \lambda(\Theta))$ with

$$\Delta t^* \lambda(\Theta) = -\Delta t^* - \frac{\nu \Delta t^*}{\Delta x}(1 - e^{-i\Theta}).$$

Possible parameters of the smoother are the pseudo time step size Δt^* and the coefficients of the RK method. Now, $\nu = a\Delta t$ is fixed during the multigrid iteration, but Δx is not. Furthermore, the pseudo time step is restricted by a CFL condition based on ν. Thus, instead of optimizing for Δt^*, we define the pseudo time step on each grid level as

$$\Delta t_l^* = c\Delta x_l / \nu$$

and optimize for $c := \nu \Delta t_l^* / \Delta x_l$. Now we have

$$z(\Theta, c; \nu, \Delta x_l) := \Delta t^* \lambda(\Theta) = -c\Delta x_l/\nu - c + ce^{-i\Theta}, \tag{5.30}$$

where we see that z does not depend on ν and Δx_l separately, but only on $\Delta x_l/\nu = 1/CFL$. Thus, with $e^{-i\Theta} = \cos(\Theta) - i\sin(\Theta)$ we obtain

$$z(\Theta, c; CFL) = -c/CFL - c + c\cos(\Theta) - ic\sin(\Theta).$$

In the end, given CFL, we have to solve an optimization problem where we look at the modulus of the maximal value of the smoother for $|\Theta| \in [\pi/2, \pi]$ and then minimize that over the parameters α_j and c. Using symmetry of P_s and equivalenty looking at the square of the modulus, we obtain

$$\min_{c, P_s} \max_{|\Theta| \in [\pi/2, \pi]} |P_s(z(\Theta, c; CFL))|^2. \tag{5.31}$$

Due to the dependence of the optimal coefficients on CFL, there is no unique optimal smoother for all problems.

For the 2-stage scheme we have:

$$\begin{aligned} |P_2(z)|^2 &= |1 + z + \alpha_1 z^2|^2 \\ &= (1 + \mathrm{Re}z + \alpha_1 \mathrm{Re}z^2 - \alpha_1 \mathrm{Im}z^2)^2 + (2\alpha_1 \mathrm{Re}z\mathrm{Im}z + \mathrm{Im}z)^2. \end{aligned}$$

Similar computations for the 3-stage scheme lead to

$$\begin{aligned}|P_3(z)|^2 =&(1+\mathrm{Re}z+\alpha_2\mathrm{Re}z^2-\alpha_2\mathrm{Im}z^2+\alpha_1\alpha_2\mathrm{Re}z^3-3\alpha_1\alpha_2\mathrm{Re}z\mathrm{Im}z^2)^2+\\ &(\mathrm{Im}z+2\alpha_2\mathrm{Re}z\mathrm{Im}z-\alpha_1\alpha_2\mathrm{Im}z^3+3\alpha_1\alpha_2\mathrm{Re}z^2\mathrm{Im}z)^2\end{aligned}$$

and for the 4-stage scheme we obtain

$$\begin{aligned}|P_4(z)|^2 = &\big(1+\mathrm{Re}z+\alpha_1\mathrm{Re}z^2-\alpha_1\mathrm{Im}z^2+\alpha_1\alpha_2\mathrm{Re}z^3-3\alpha_1\alpha_2\mathrm{Re}z\mathrm{Im}z^2\\ &+\alpha_1\alpha_2\alpha_3\mathrm{Re}z^4-6\alpha_1\alpha_2\alpha_3\mathrm{Re}z^2\mathrm{Im}z^2+\alpha_1\alpha_2\alpha_3\mathrm{Im}z^4\big)^2+\\ &\big(\mathrm{Im}z+2\alpha_1\mathrm{Re}z\mathrm{Im}z+3\alpha_1\alpha_2\mathrm{Re}z^2\mathrm{Im}z-\alpha_1\alpha_2\mathrm{Im}z^3\\ &+4\alpha_1\alpha_2\alpha_3\mathrm{Re}z^3\mathrm{Im}z-4\alpha_1\alpha_2\alpha_3\mathrm{Re}z\mathrm{Im}z^3\big)^2.\end{aligned}$$

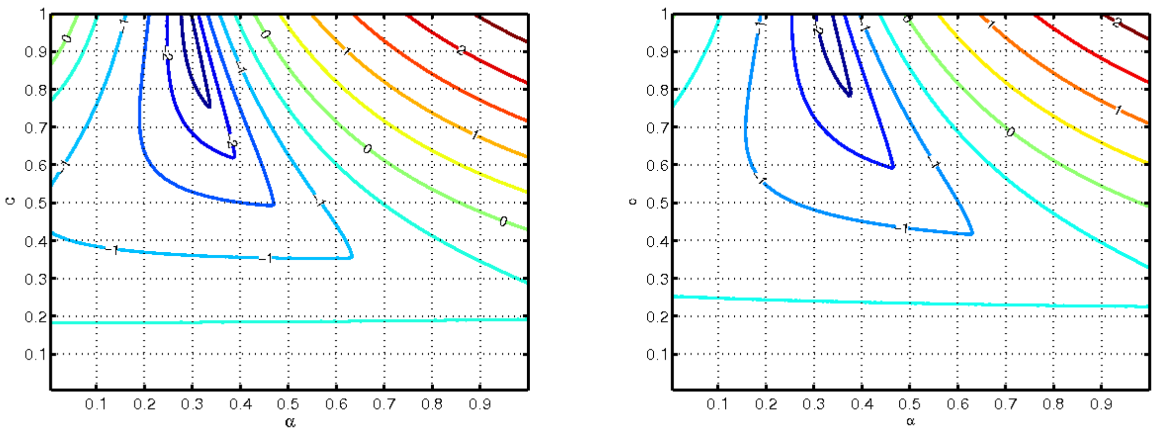


Figure 5.6: Contourplots of functions $\log_{10}\max_{|\Theta|\in[\pi/2,\pi]}|P_2(z(\Theta,c;3))|^2$ (left) and $\log_{10}\max_{|\Theta|\in[\pi/2,\pi]}|P_2(z(\Theta,c;24))|^2$ (right).

It turns out that for these functions, the final form of (5.31) is too difficult to solve exactly, in particular due to the min-max-formulation. Therefore, we discretize the parameter space and compute an approximate solution. This requires a bounded region, which is already the case for Θ and the α_j, which are between 0 and 1. As for c, we know that any explicit RK scheme has a bounded stability region, therefore we chose an upper bound for c, such that the optimal value for c is not on the boundary. As an example, the function

$$f(\alpha,c) := \log_{10}\max_{|\Theta|\in[\pi/2,\pi]}|P_2(z(\Theta,c;CFL))|^2$$

is shown in figure 5.6 for $CFL = 3$ and $CFL = 24$. Note that the optimal c is not on the boundary, meaning that the choice $c \in [0,1]$ here is reasonable. Furthermore, we can see that for $c = 0$, we obtain a method with a value of 1. This is correct, since this is a method

CFL	α	c	Opt-value
1	0.275	0.745	0.01923
3	0.3	0.93	0.05888
6	0.315	0.96	0.08011
9	0.32	0.975	0.08954
12	0.325	0.97	0.09453
24	0.33	0.98	0.10257

Table 5.2: Results of optimization of smoothing properties for 2-stage scheme

CFL	α_1	α_2	c	Opt-value
1	0.12	0.35	1.14	0.001615
3	0.135	0.375	1.37	0.007773
6	0.14	0.385	1.445	0.01233
9	0.14	0.39	1.45	0.01486
12	0.145	0.395	1.44	0.01588
24	0.145	0.395	1.495	0.01772

Table 5.3: Results of optimization of smoothing properties for 3-stage scheme

with time step zero, meaning that the resulting smoother is the identity. For $\alpha = 0$, we obtain the explicit Euler method. This is also a possible smoother, but as can be seen it is less powerful. Furthermore, we can see the finite stability regions of the methods.

We now compute the optimal value for $CFL = 1, 3, 6, 9, 12, 24$ for all schemes using a MATLAB/C++ code. For the 2-stage scheme, we choose a grid of $200 \times 200 \times 200$ for the parameter space $\alpha_1 \times c \times t$. The optimization gives results presented in table 5.2. As can be seen, $CFL = 1$ is a special case, otherwise the parameter α_1 does not depend on CFL, whereas there is a slight increase of c with CFL.

For the 3-stage scheme, we have one more parameter, which increases the dimension of the parameter space $\alpha_1 \times \alpha_2 \times c \times t$, resulting in the grid $200 \times 200 \times (2 \cdot 200) \times 200$. As a restriction for c, we put $c \in [0, 2]$. The results can be seen in table 5.3. Again, $CFL = 1$ is a special case in that a significantly smaller value for c comes out. Otherwise, the coefficients α_1 and α_2 have only a weak dependence on CFL. However, the optimal value is decreased by a factor of about 500, suggesting that these schemes are significantly better than the 2-stage methods.

Finally, for the 4-stage scheme, we chose a grid of the size $150 \times 150 \times 150 \times (2 \cdot 150) \times 100$

CFL	α_1	α_2	α_3	c	Opt-value
1	0.0733	0.1867	0.4	1.3267	0.0005302
3	0.0733	0.1867	0.4	1.96	0.002118
6	0.08	0.2	0.4133	1.907	0.00297
9	0.08	0.2	0.4133	1.98	0.003372
12	0.08	0.2	0.42	1.907	0.003972
24	0.0867	0.2133	0.433	1.84	0.004313

Table 5.4: Results of optimization of smoothing properties for 4-stage scheme

with $c \in [0, 2]$. A finer grid was not possible due to storage restrictions. As for the 3-stage case, a significantly smaller value for c is obtained for $CFL = 1$, otherwise there is only a weak dependence of the parameters on CFL. The optimal value is decreased by a factor of four compared to the 3-stage schemes, as shown in table 5.4.

An important difference between the schemes obtained is the size of c, which is about 0.9 for the 2-stage case, 1.4 for the 3-stage case and 1.9 for the 4-stage case, which suggests that one effect of allowing more freedom in the design of the smoother is that the stability region is increased. The stability regions of the optimal methods obtained for $CFL = 3$ and $CFL = 24$ are shown in figure 5.7, emphasizing this point.

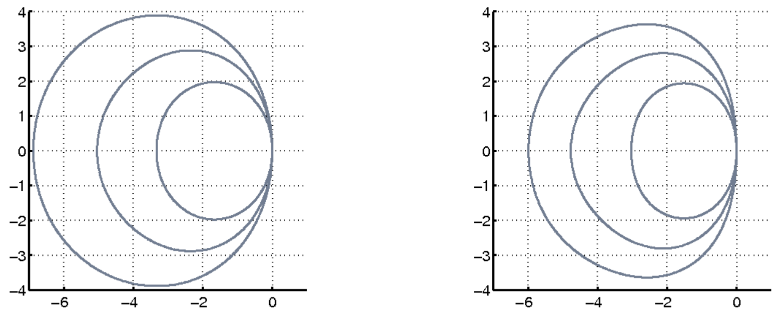


Figure 5.7: Stability regions of optimally smoothing methods for $CFL = 3$ (left) and $CFL = 24$ (right). The larger the stability region, the higher the number of stages.

5.6.8 Optimizing the spectral radius

The optimization just considered aims at improving the smoother on its own, without taking into account the interaction with the coarse grid correction or the multigrid structure. This has the benefit that the optimization is fast, even for the 4-stage case, where we run into memory problems. An alternative is to optimize the spectral radius of the iteration matrix $\mathbf{M}$ of the complete multigrid scheme as a function of the smoother, which is a function of α and c, with c defined as above:

$$\min_{\alpha,c} \rho(\mathbf{M}(\alpha, c; \nu, \Delta x)). \tag{5.32}$$

Regarding the iteration matrix, it is important to note that the smoother is furthermore a function of a right hand side and an approximate solution.

Thus, when applying the s-stage Runge-Kutta smoother on level l to an ODE with right hand side $\mathbf{f}_l(\mathbf{u}) = \mathbf{b} - \mathbf{A}_l\mathbf{u}$, there is an additional term on top of the stability polynomial, depending on the vector $\mathbf{b}$:

$$\mathbf{S}_{s,l}(\mathbf{b}, \mathbf{u}) = \mathbf{S}^u_{s,l}\mathbf{u} + \mathbf{S}^b_{s,l}\mathbf{b},$$

Here, $\mathbf{S}^u_{s,l} = P_s(-\Delta t^*\mathbf{A}_l)$ is the matrix coming out of the stability polynomial, whereas the second matrix corresponds to a different polynomial. For the 2-, 3- and 4-stage smoother, we have

$$\mathbf{S}^b_{s,l} = \Delta t\mathbf{I}_l - \alpha_1\Delta t^2\mathbf{A}_l, \tag{5.33}$$
$$\mathbf{S}^b_{3,l} = \Delta t\mathbf{I}_l - \alpha_1\Delta t^2\mathbf{A}_l + \alpha_1\alpha_2\Delta t^3\mathbf{A}_l^2, \tag{5.34}$$
$$\mathbf{S}^b_{4,l} = \Delta t\mathbf{I}_l - \alpha_1\Delta t^2\mathbf{A}_l + \alpha_1\alpha_2\Delta t^3\mathbf{A}_l^2 - \alpha_1\alpha_2\alpha_3\Delta t^4\mathbf{A}_l^3. \tag{5.35}$$

When first calling the multigrid function, this is done with the actual right hand side $\mathbf{b}$ and the current approximation $\mathbf{u}^{(k)}$. However, on lower levels the right hand side is the restricted residual $\mathbf{R}_{l-1,l}\mathbf{r}_l$ and the current approximation is the zero vector. For a three-level scheme, we have

$$\begin{aligned}\mathbf{u}^{(k+1)} &= \mathbf{S}^u_{s,2}\mathbf{u}^{(k)} + \mathbf{S}^b_{s,2}\mathbf{b} + \mathbf{P}_{2,1}\mathbf{MG}(\mathbf{0}, \mathbf{r}_1, 1)\\ &= \mathbf{S}^u_{s,2}\mathbf{u}^{(k)} + \mathbf{S}^b_{s,2}\mathbf{b} + \mathbf{P}_{2,1}(\mathbf{S}^b_{s,1}\mathbf{r}_1 + \mathbf{P}_{1,0}\mathbf{S}^b_{s,0}\mathbf{r}_0),\end{aligned} \tag{5.36}$$

where

$$\mathbf{r}_0 = \mathbf{R}_{0,1}(\mathbf{I}_1 - \mathbf{A}_1\mathbf{S}^b_{s,1})\mathbf{r}_1 \tag{5.37}$$

and

$$\begin{aligned}\mathbf{r}_1 &= \mathbf{R}_{1,2}(\mathbf{b} - \mathbf{A}_2(\mathbf{S}^b_{s,2}\mathbf{b} + \mathbf{S}^u_{s,2}\mathbf{u}^{(k)}))\\ &= \mathbf{R}_{1,2}(\mathbf{I}_2 - \mathbf{A}_2\mathbf{S}^b_{s,2})\mathbf{b} - \mathbf{R}_{1,2}\mathbf{A}_2\mathbf{S}^u_{s,2}\mathbf{u}^{(k)}.\end{aligned} \tag{5.38}$$

Inserting (5.37) and (5.38) into (5.36), we obtain

$$\begin{aligned}\mathbf{u}^{(k+1)} &= \mathbf{S}^u_{s,2}\mathbf{u}^{(k)} + \mathbf{S}^b_{s,2}\mathbf{b} + \mathbf{P}_{2,1}(\mathbf{S}^b_{s,1} + \mathbf{P}_{1,0}\mathbf{S}^b_{s,0}\mathbf{R}_{0,1}(\mathbf{I}_1 - \mathbf{A}_1\mathbf{S}^b_{s,1}))\mathbf{r}_1\\ &= (\mathbf{S}^b_{s,2} + \mathbf{P}_{2,1}(\mathbf{S}^b_{s,1} + \mathbf{P}_{1,0}\mathbf{S}^b_{s,0}\mathbf{R}_{0,1}(\mathbf{I}_1 - \mathbf{A}_1\mathbf{S}^b_{s,1}))\mathbf{R}_{1,2}(\mathbf{I}_2 - \mathbf{A}_2\mathbf{S}^b_{s,2}))\mathbf{b}\\ &+ (\mathbf{S}^u_{s,2} - \mathbf{P}_{2,1}(\mathbf{S}^b_{s,1} + \mathbf{P}_{1,0}\mathbf{S}^b_{s,0}\mathbf{R}_{0,1}(\mathbf{I}_1 - \mathbf{A}_1\mathbf{S}^b_{s,1}))\mathbf{R}_{1,2}\mathbf{A}_2\mathbf{S}^u_{s,2})\mathbf{u}^{(k)}.\end{aligned}$$

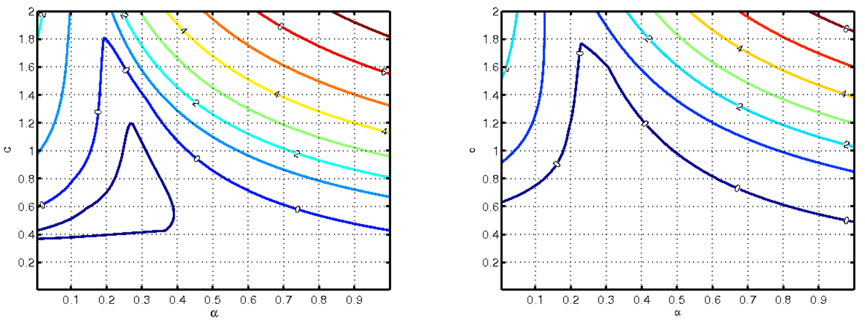


Figure 5.8: 2-stage method: Contourplots of $\log(\rho(\mathbf{M}(\alpha, c)))$ for $\Delta x = 1/24$ and $\nu = 0.125$ (left) and $\nu = 1.0$ (right).

Thus, the iteration matrix of the three level scheme is given by

$$\mathbf{M} = \mathbf{S}^u_{s,2} - \mathbf{P}_{2,1}(\mathbf{S}^b_{s,1} + \mathbf{P}_{1,0}\mathbf{S}^b_{s,0}\mathbf{R}_{0,1}(\mathbf{I}_1 - \mathbf{A}_1\mathbf{S}^b_{s,1}))\mathbf{R}_{1,2}\mathbf{A}_2\mathbf{S}^u_{s,2}.$$

To solve the optimization problem (5.32), we again compute a discrete optimum, this time using a MATLAB code and the eig function to obtain the spectral radius. For the 2-stage case, we use a grid of size $200 \times (2 \cdot 200)$ with $c \in [0, 2]$. Here, both ν and Δx have to be varied, where we chose ν according to the same CFL numbers as for the last strategy and $\Delta x = 1/24, 1/12, 1/6$. The results are shown in table 5.5. The contour lines of the function $\rho(\mathbf{M}(\alpha, c; \nu, \Delta x))$ are illustrated in figure (5.8). As can be seen, the results are qualitatively similar to the ones from the other strategy.

In the 3-stage case, we use a grid of the size $100 \times 100 \times (2 \cdot 100)$, the results are shown in table 5.6. Finally, a $50 \times 50 \times 50 \times (2 \cdot 50)$ grid is used for the 4-stage case with the results being shown in table 5.7. The decrease in mesh width is due to the polynomially growing computational cost. One optimization on the fine grid in the 2-stage case takes a couple of minutes, whereas the 4-stage case needs more than ten hours, despite the coarser grid.

Comparing the results for the different stages, we see that there is no dependence of the optimal solution on Δx in the sense that for a fixed CFL number and accordingly chosen ν, the results are almost identical. In this sense, the multigrid method obtained has a convergence speed which is independent of the mesh width. Otherwise, an increase in CFL number leads to an increase in the spectral radius and thus a decrease of the convergence speed of the methods. Regarding the coefficients, there is a weak dependence of α on the CFL number, but c increases significantly with CFL. Regarding the size of the spectral radius, going from two to three stages leads to a huge decrease of the optimal value for small and moderate CFL number, but not for large CFL numbers. As for adding a fourth stage, this actually leads to a decrease in spectral radius. This can be explained by the much coarser grid used for the optimization of the 4-stage method. Consequently, the solution found is too far away from the actual optimum to beat the 3-stage method.

	$\Delta x = 1/24$				$\Delta x = 1/12$			
CFL	ν	α	c	$\rho(\mathbf{M})$	ν	α	c	$\rho(\mathbf{M})$
1	1/24	0.25	0.545	0.0689	1/12	0.25	0.56	0.0681
3	0.125	0.21	0.615	0.2072	0.25	0.21	0.615	0.2072
6	0.25	0.305	0.94	0.3007	0.5	0.315	0.96	0.2954
9	0.375	0.295	1.145	0.3824	0.75	0.3	1.145	0.3819
12	0.5	0.295	1.255	0.4584	1.0	0.3	1.26	0.4575
24	1.0	0.295	1.46	0.6425	2.0	0.29	1.485	0.6371
	$\Delta x = 1/6$							
1	1/6	0.245	0.555	0.0673				
3	0.5	0.27	0.77	0.1851				
6	1.0	0.295	1.0	0.2734				
9	1.5	0.29	1.175	0.3694				
12	2.0	0.29	1.29	0.4473				
24	4.0	0.28	1.51	0.6315				

Table 5.5: Results of optimization of $\rho(\mathbf{M})$ for 2-stage scheme

	$\Delta x = 1/24$					$\Delta x = 1/12$				
CFL	ν	α_1	α_2	c	$\rho(\mathbf{M})$	ν	α_1	α_2	c	$\rho(\mathbf{M})$
1	1/24	0.11	0.33	0.69	0.0402	1/12	0.11	0.33	0.69	0.0402
3	0.125	0.14	0.39	1.42	0.0819	0.25	0.14	0.39	1.43	0.0799
6	0.25	0.14	0.4	1.58	0.1444	0.5	0.14	0.4	1.6	0.1397
9	0.375	0.12	0.37	1.75	0.2317	0.75	0.12	0.36	1.77	0.2237
12	0.5	0.13	0.39	1.91	0.3124	1.0	0.12	0.36	1.95	0.2954
24	1.0	0.12	0.38	2.14	0.5252	2.0	0.10	0.31	2.42	0.4720
	$\Delta x = 1/6$									
1	1/6	0.11	0.33	0.68	0.0381					
3	0.5	0.14	0.39	1.43	0.0799					
6	1.0	0.13	0.38	1.67	0.1375					
9	1.5	0.12	0.36	1.78	0.2230					
12	2.0	0.11	0.34	1.93	0.2948					
24	4.0	0.09	0.28	2.59	0.4427					

Table 5.6: Results of optimization of $\rho(\mathbf{M})$ for 3-stage scheme

	$\Delta x = 1/24$						$\Delta x = 1/12$					
CFL	ν	α_1	α_2	α_3	c	$\rho(\mathbf{M})$	ν	α_1	α_2	α_3	c	$\rho(\mathbf{M})$
1	1/24	0.02	0.08	0.3	0.62	0.0525	1/12	0.02	0.08	0.3	0.62	0.0525
3	0.125	0.02	0.14	0.38	1.42	0.1138	0.25	0.02	0.14	0.38	1.42	0.1138
6	0.25	0.02	0.14	0.38	1.52	0.1783	0.5	0.02	0.14	0.38	1.52	0.1783
9	0.375	0.02	0.14	0.4	1.7	0.2501	0.75	0.02	0.12	0.36	1.78	0.2365
12	0.5	0.02	0.12	0.36	1.9	0.3053	1.0	0.02	0.12	0.34	1.88	0.3040
24	1.0	0.02	0.12	0.34	2.16	0.5173	2.0	0.02	0.1	0.30	2.18	0.5094
	$\Delta x = 1/6$											
1	1/6	0.02	0.08	0.3	0.62	0.0492						
3	0.5	0.02	0.14	0.38	1.42	0.1138						
6	1.0	0.02	0.14	0.38	1.52	0.1783						
9	1.5	0.02	0.12	0.36	1.78	0.2236						
12	2.0	0.02	0.12	0.34	1.88	0.3040						
24	4.0	0.02	0.10	0.32	2.34	0.4858						

Table 5.7: Results of optimization of $\rho(\mathbf{M})$ for 4-stage scheme

In figure 5.9, the stability region of the optimal methods for $\Delta x = 1/24$ and $\nu = 0.125$, as well as $\nu = 1.0$ are shown. Again, an increase in the number of stages leads to a larger stability region.

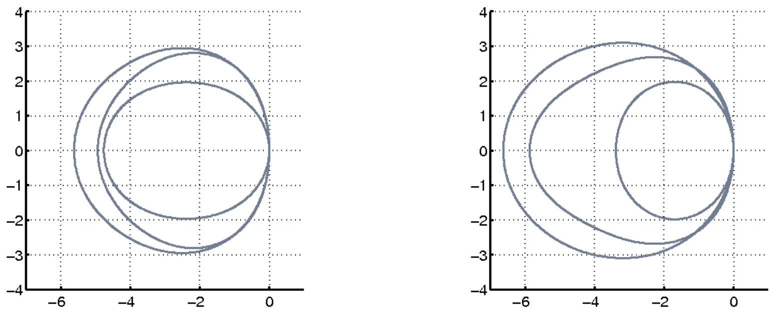


Figure 5.9: Stability region of optimal 2-, 3- and 4-stage method for $\Delta x = 1/24$ and $\nu = 0.125$ (left) and $\nu = 1.0$ (right). The larger the number of stages, the larger the stability region.

Comparing the optimal solutions found with the schemes obtained by the previous method, we see that the coefficients are similar, as is the value of c and the same is valid for the stability regions.

5.6.9 Numerical results

We test the smoothers on two problems with $\Delta x = 1/24$ on the finest level and $a = 2.0$. As initial conditions, we use a step function with values 5 and 1, as well as the function $\sin(\pi x)$. We then perform one time step with $\Delta t = 1/16$, respectively $\Delta t = 0.5$, meaning that $\nu = 0.125$ and $CFL = 3$ (a problem with a medium CFL number), respectively $\nu = 1.0$ and $CFL = 24$ (a problem with a large CFL number). The resulting linear equation system is solved with 80 steps of the different multigrid methods. As a reference, the optimal 2- and 3-stage methods derived by van Leer et. al. [208] for steady state problems are used. The 2-stage method is given by $\alpha = 1/3$ and a $c = 1$, whereas the 3-stage method does not actually fall into the framework considered here. It consists of one step of the explicit Euler method with $c = 0.5$, followed by a step of a 2-stage method with $\alpha = 0.4$ and $c = 1$. All computations are performed using MATLAB.

In figure 5.10, the computed solutions for the linear advection equation are shown. The initial data is always shown in blue, whereas the exact solution is in red, as well as the numerical one. Since the space discretization is of first order and the time integration method is implicit Euler, the results are very diffusive. However, the multigrid method computes the correct solution.

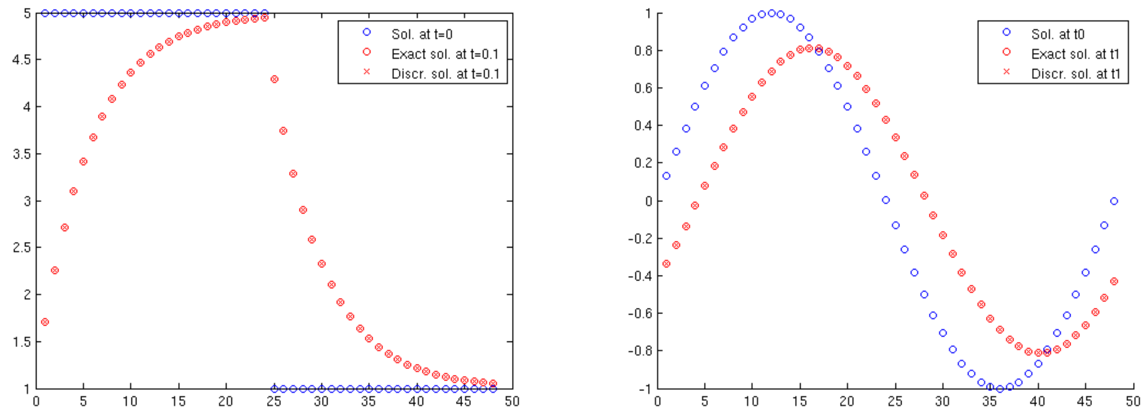


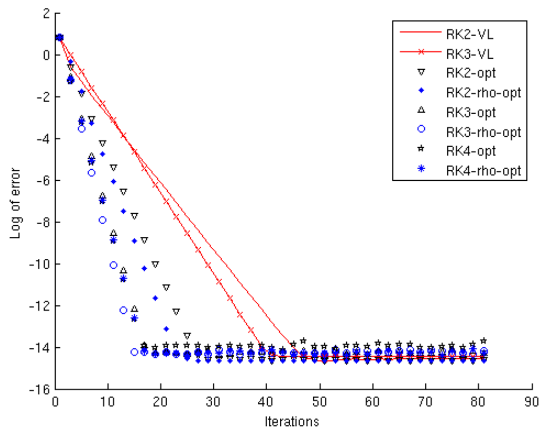
Figure 5.10: Initial solution and discrete and numerical solution after one time step for step function (left) and sine initial data (right).

Regarding computational effort, the main work of the multigrid method consists of matrix vector multiplications in the smoother. Thus, the 3-stage schemes are conservatively 50% more costly than the 2-stage schemes, whereas the 4-stage schemes are less than twice as expensive as the RK-2 smoother.

We now look at the convergence speed of the different methods, where we call the methods obtained by the second optimization ρ-optimized schemes. In figure 5.11 and 5.12, $\log_{10}$ of the error in the 2-norm is plotted over multigrid iterations, where the first figure shows the results for $CFL = 3$ for both test cases and the latter the results for $CFL = 24$ for both test cases. The 3-stage method of van Leer diverges for $CFL = 24$ and is only shown in the first figure, where it is barely faster than the 2-stage method of van Leer. Otherwise we can see, that the ρ-optimized schemes behave as expected in that the 3-stage scheme is the fastest, then the 4-stage scheme and then the 2-stage scheme with the 3-stage scheme being roughly twice as fast as the 2-stage scheme. For the schemes coming out of the first optimization, there the 4-stage scheme is faster than the 3-stage scheme, which is faster than the 2-stage scheme. Furthermore, the ρ-optimized schemes are able to beat their counterparts with the exception of the 4-stage scheme. Thus, the more costly optimizition is generally worthwhile.

Generally, the 3-stage ρ-optimized scheme is the fastest, in particular it is almost twice as fast as the 2-stage ρ-optimized scheme, making it more efficient. Compared to the reference method of van Leer, it is between two and four times faster, making it between 70% and 270% more efficient. Thus, just by changing the coefficients of the RK smoother, we can expect to gain more than a factor of two in multigrid efficiency.

Finally, we consider the step function case with nonperiodic boundary, to evaluate if the different eigenvalues respectively different matrices lead to problems. As can be seen in figure 5.13, this is not the case for $CFL = 3$, as the convergence rate for all methods



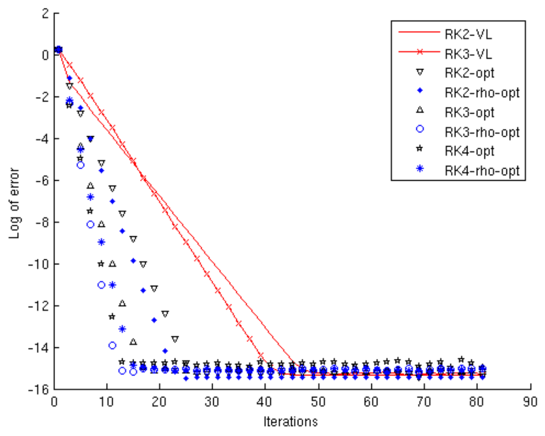


Figure 5.11: Convergence plots for different multigrid methods and $CFL = 3$: step function (left) and sine initial data (right).

is almost unchanged, but not so for $CFL = 24$, where the new methods have the same convergence speed, but van Leers 2-stage method becomes as fast as the ρ-optimized 2-stage method.

5.7 Newton's method

A classical method to solve nonlinear equation systems is Newton's method, sometimes referred to as the Newton-Raphson method. The basic idea is to linearize the problem and thus replace the nonlinear problem by a sequence of linear problems. For the one dimensionsal case, the method is illustrated in figure 5.14. Newton's method is locally convergent and can exhibit quadratic convergence. Outside the region of convergence, it is not clear what happens. The method can converge nevertheless or if equation (5.39) has multiple solutions, it may converge to a different solution, otherwise, it diverges. In fact, it has been shown for a number of equations that the set of starting points that converge to a certain solution is fractal.

The local convergence means that its use for steady flows is limited, since the typical starting guess in that case consists of choosing freestream values in the whole domain, which are far away from the steady state values. This problem can be adressed by globalization strategies, but while this leads to a convergent method, it still takes very long to find a good approximation. For unsteady problems, the situation is different in that the solution at time t_n is not that far away from the solution at t_{n+1}. Therefore, Newton's method has to be reassessed in that situation. Jothiprasad et. al. compare the FAS scheme with an inexact Newton method where either linear multigrid or GMRES is used as a solver and find the FAS scheme to be computationally the worst [103].

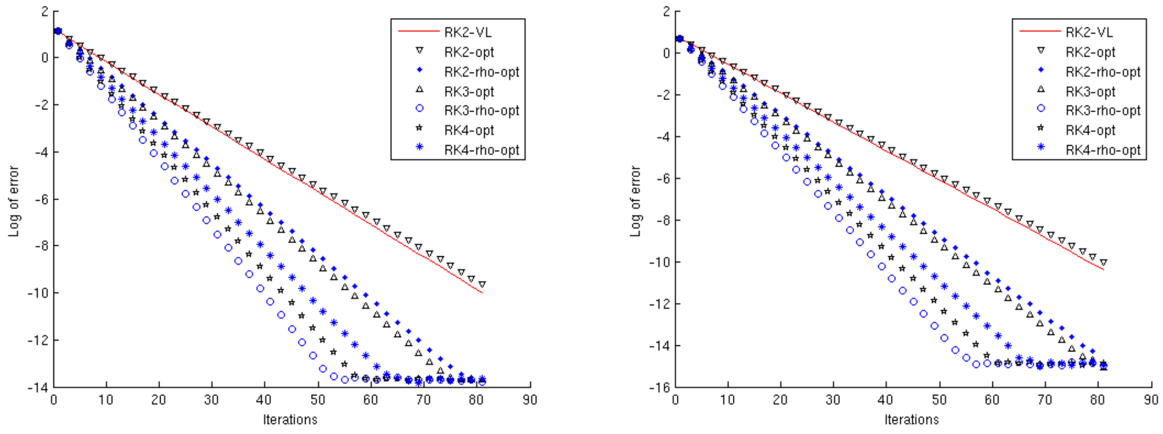


Figure 5.12: Convergence plots for different multigrid methods and $CFL = 24$: step function (left) and sine initial data (right).

We will now explain those parts of the theory relevant to the methodology used here and otherwise refer to the books of Kelley [108] and Deuflhard [51]. Newton's method solves the root problem

$$\underline{\mathbf{F}}(\underline{\mathbf{u}}) = \underline{\mathbf{0}} \tag{5.39}$$

for a differentiable function $\underline{\mathbf{F}}(\underline{\mathbf{u}})$ in the following way:

$$\begin{aligned} \left.\frac{\partial \mathbf{F}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}\right|_{\underline{\mathbf{u}}^{(k)}} \Delta\underline{\mathbf{u}} &= -\mathbf{F}(\underline{\mathbf{u}}^{(k)}) \\ \underline{\mathbf{u}}^{(k+1)} &= \underline{\mathbf{u}}^{(k)} + \Delta\underline{\mathbf{u}}, \quad k = 0, 1, \ldots. \end{aligned} \tag{5.40}$$

The method must not necessarily be written in conservative variables. Instead, derivatives of primitive variables could be used without any difference in convergence speed. As termination criteria, either (5.8) or (5.9) is used.

Convergence of the method can be proved under very few assumptions. However, a major point about Newton's method is that it is second order convergent. To prove that, the following standard assumptions are used in [108]:

Definition 8 (3.2: Standard Assumptions)

i) Equation (5.39) has a solution $\underline{\mathbf{u}}^*$.

ii) $\underline{\mathbf{F}}' : \Omega \rightarrow \mathbb{R}^{m \times m}$ *is Lipschitz continuous with Lipschitz constant* L'.

iii) $\underline{\mathbf{F}}'(\underline{\mathbf{u}}^*)$ *is nonsingular.*

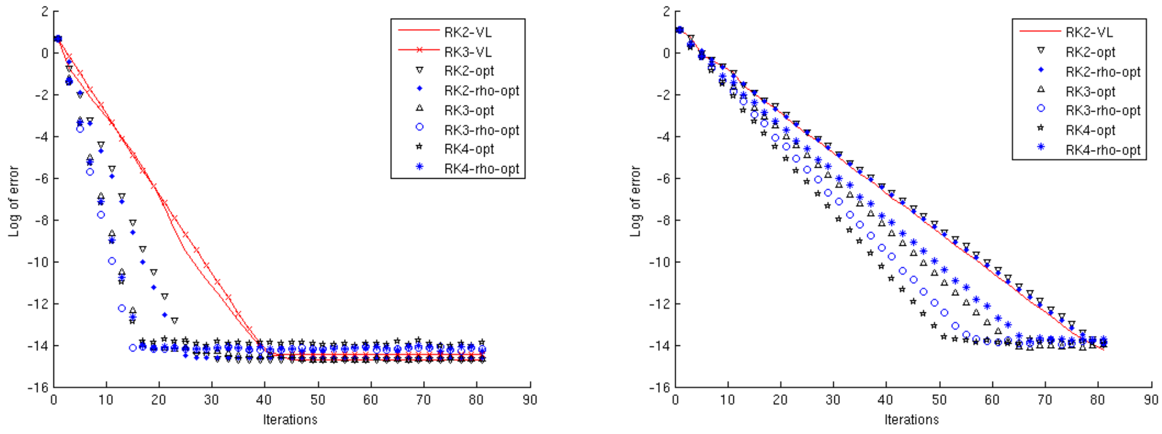


Figure 5.13: Convergence plots for different multigrid methods for step function initial data with nonperiodic boundary conditions: CFL 3 (left) and CFL 24 (right).

Note: Since the determinant is a continuous function, iii) implies with ii) that F' is non-singular in a whole neighborhood around x^* and thus, Newton's method is well defined in that neighborhood.

If the linear equation systems (5.40) are solved exactly, the method is locally second order convergent. However, this is never done, because an exact Jacobian is rarely available and furthermore, this is too costly and unnecessary. Instead, we will approximate terms in (5.40) and solve the linear systems only approximately. Thus we might lose convergence speed, but we save a significant amount of CPU time. Since in case of convergence, the limit is determined by the right hand side, we can approximate the matrix or $\Delta\underline{\mathbf{u}}$ without disturbing the outer time integration scheme.

Simplified Newton's method: If we approximate the matrix, there is first of all the simplified Newton's method, where we compute the Jacobian only once and thus iterate:

$$\begin{aligned} \left.\frac{\partial \mathbf{F}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}\right|_{\underline{\mathbf{u}}^{(0)}} \Delta\underline{\mathbf{u}} &= -\mathbf{F}(\underline{\mathbf{u}}^{(k)}) \\ \underline{\mathbf{u}}^{(k+1)} &= \underline{\mathbf{u}}^{(k)} + \Delta\underline{\mathbf{u}}, \quad k = 0, 1, ... \end{aligned} \tag{5.41}$$

This method is also locally convergent and has convergence order one. It is frequently used in a variant, where the Jacobian is recomputed periodically, see e.g. [**?**]. Due to the high cost of computing the Jacobian, the loss of order of convergence is typically less important.

Methods of Newton type: The simplified Newton method can be generalized to so called methods of Newton type, where we approximate the Jacobian by some matrix $\mathbf{A}$. For

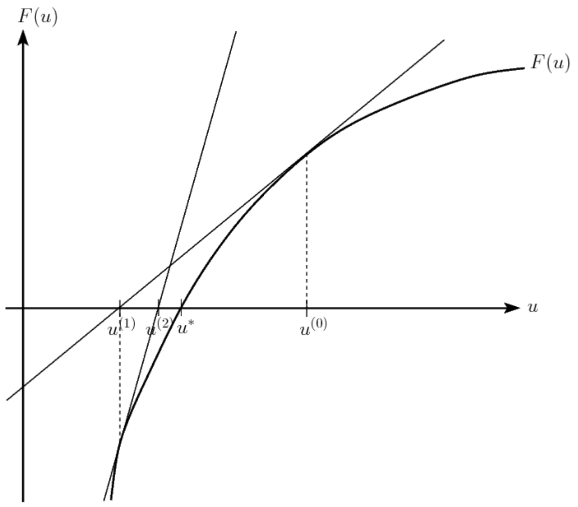


Figure 5.14: Illustration of Newton's method in one dimension

example, this could correspond to a lower order Jacobian for a higher order discretization. We thus obtain the scheme:

$$\begin{aligned} \mathbf{A}\Delta\underline{\mathbf{u}} &= -\mathbf{F}(\underline{\mathbf{u}}^{(k)}) \\ \underline{\mathbf{u}}^{(k+1)} &= \underline{\mathbf{u}}^{(k)} + \Delta\underline{\mathbf{u}}, \quad k = 0, 1, ... \end{aligned} \tag{5.42}$$

This method converges locally linear, if $\mathbf{A}$ is close enough to the Jacobian or more precise, if $\rho(I - \mathbf{A}^{-1}\frac{\partial \mathbf{F}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}) < 1$, respectively if $\|\mathbf{A} - \frac{\partial \mathbf{F}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}\|$ is small enough. In the proof, the size of the neighborhood of the solution where convergence can be proved depends on the spectral radius just mentioned in the sense that it becomes smaller, the further away $\mathbf{A}$ is from $\frac{\partial \mathbf{F}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}$.

Inexact Newton methods: Another approach to saving computing time are inexact Newton methods. There, the linear equation systems are solved by an iterative scheme, for example one of the later discussed Krylov subspace methods. These are terminated prematurely, based on the residual of the linear equation system. Typically, the scheme suggested in [49] is chosen, where the inner solver is terminated, if the relative residual is below a certain threshhold. This type of scheme can be written as:

$$\begin{aligned} \left\| \frac{\partial \mathbf{F}(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}}|_{\underline{\mathbf{u}}^{(k)}} \Delta\underline{\mathbf{u}} + \mathbf{F}(\underline{\mathbf{u}}^{(k)}) \right\| &\leq \eta_k \|\mathbf{F}(\underline{\mathbf{u}}_k)\| \\ \underline{\mathbf{u}}^{(k+1)} &= \underline{\mathbf{u}}^{(k)} + \Delta\underline{\mathbf{u}}, \quad k = 0, 1, ... \end{aligned} \tag{5.43}$$

The $\eta_k \in \mathbb{R}$ are called forcing terms. In [56], the choice for this sequence is discussed and the following theorem is proved:

Theorem 6 *Let the standard assumptions hold. Then there is δ such that if $\underline{\mathbf{u}}^{(0)}$ is in a δ-neighborhood of $\underline{\mathbf{u}}^*$, $\{\eta_k\} \subset [0, \eta]$ with $\eta < \bar{\eta} < 1$, then the inexact Newton iteration (5.43) converges linearly. Moreover,*

- *if $\eta_k \to 0$, the convergence is superlinear and*
- *if $\eta_k \leq K_\eta \|\mathbf{F}(\underline{\mathbf{u}}^{(k)})\|^p$ for some $K_\eta > 0$ and $p \in [0, 1]$, the convergence is superlinear with order $1 + p$.*

The region of convergence given by δ is smaller or as large as the one for the exact Newton's method. Furthermore, the convergence region decreases with growth of $\kappa(\mathbf{F}'(\mathbf{u}))$. Regarding speed of convergence, it follows from the theorem that the common strategy of choosing a fixed η leads to a first order convergence speed. The last part of the theorem means that for a properly chosen sequence of forcing terms, namely, if the η_k converge to zero fast enough, convergence is quadratic. However, it is not necessary to solve the first few linear systems very accurately. This is in line with the intuition, that while we are far away from the solution, we do not need the optimal search direction for Newton's method, but just a reasonable one, to get us in the generally correct direction.

A way of achieving this is the following:

$$\eta_k^A = \gamma \frac{\|\mathbf{F}(\underline{\mathbf{u}}^{(k)})\|^2}{\|\mathbf{F}(\underline{\mathbf{u}}^{(k-1)})\|^2}$$

with a parameter $\gamma \in (0, 1]$. This was suggested by Eisenstat and Walker in [56], where they also prove that this sequence has the convergence behavior required for the theorem. Thus, the theorem says that if this sequence is bounded away from one uniformly, convergence is quadratic. Therefore, we set $\eta_0 = \eta_{max}$ for some $\eta_{max} < 1$ and for $k > 0$:

$$\eta_k^B = \min(\eta_{max}, \eta_k^A).$$

Eisenstat and Walker furthermore suggest safeguards to avoid volatile decreases in η_k. To this end, $\gamma\eta_{k-1}^2 > 0.1$ is used as a condition to determine if η_{k-1} is rather large and thus the definition of η_k is refined to

$$\eta_k^C = \begin{cases} \eta_{max}, & n = 0, \\ \min(\eta_{max}, \eta_k^A), & n > 0, \gamma\eta_{k-1}^2 \leq 0.1 \\ \min(\eta_{max}, \max(\eta_k^A, \gamma\eta_{k-1}^2)) & n > 0, \gamma\eta_{k-1}^2 > 0.1 \end{cases}$$

Finally, to avoid oversolving in the final stages, Eisenstat and Walker suggest

$$\eta_k = \min(\eta_{max}, \max(\eta_k^C, 0.5\epsilon / \|\mathbf{F}(\underline{\mathbf{u}}^{(k)})\|)), \tag{5.44}$$

where ϵ is the tolerance at which the Newton iteration would terminate, see (5.8) and (5.9).

5.7.1 Choice of initial guess

The choice of the inital guess in Newton's method is important for two reasons: Firstly, the method is only locally convergent, meaning that for bad choices of initial guesses, the method may not converge. Secondly, a smart initial guess may reduce the number of iterations significantly.

A reasonable starting value for the Newton iterations for unsteady computations is the solution from the last time level, respectively the last stage value inside a DIRK scheme. For small time steps, this will be in the region of convergence, if the time step is too large, the method will diverge. Typically, this type of initial guess works for reasonably large time steps. If not, as discussed in section 5.4, we will repeat the time step with half the step size, hopefully leading to a situation where the initial guess is in the region of convergence. Nevertheless, better initial guesses can be obtained, for example by extrapolation, proper orthogonal decomposition [200] or dense output formulas (4.24). Extrapolation corresponds to taking the sequence of solutions of the previous nonlinear systems and extrapolating it in some way to the next system. Dense output formulas originate in the DIRK methods used.

5.7.2 Globally convergent Newton methods

Even with an improved initial guess obtained using any of the methods from the last section, we might not be in the region of convergence of Newton's method. This makes globally convergent variants of Newton's method attractive. This property can be achieved in a number of ways. Here, we focus on using a so called line search, which is easy to implement. This means that we change the update in the method to

$$\underline{\mathbf{u}}^{(k+1)} = \underline{\mathbf{u}}^{(k)} + \lambda \Delta \underline{\mathbf{u}}^{(k)},$$

where λ is a real number between 0 and 1. Different ways of doing this line search exist, a popular choice is the Armijo line search [50]. Here, λ is chosen such that we obtain a sequence of residuals that is almost monotone:

$$\|\mathbf{f}(\underline{\mathbf{u}}^{(k+1)}\| < (1 - \alpha\lambda)\|\mathbf{f}(\underline{\mathbf{u}}^{(k)}\|, \tag{5.45}$$

with α being a small positive number. For example, $\alpha = 10^{-4}$ is a suitable choice. The search for a suitable λ is done in a trivial way by starting with 1 and then dividing it by 2 if the condition (5.45) is not satisfied. More sophisticated strategies are possible.

It can be shown that this method is linearly convergent [50], however once the iteration is close enough to the solution, full steps with $\lambda = 1$ can be taken and thus, the original, possibly quadratic, convergence behavior is recovered.

Globally convergent Newton methods can be used to compute steady state solutions. However, this does not lead to schemes that are competitive with the FAS multigrid. A variant that is often used is to consider the backward Euler method with just one Newton step per time step and then choose a CFL number that increases with time. This corresponds to a damped Newton method, where the choice of the λ from above does not originate in a line search, but a time step chosen. In figure 5.15, the norm of the steady state residual is shown for a run of TAU_2D with linear reconstruction, AUSMDV flux and Barth limiter for a NACA0012 profile computation. The Mach number is 0.85, the Reynolds number is 1.000, the angle of attack 1.25, the computation is started with freestream values and the number of cells is 4605.

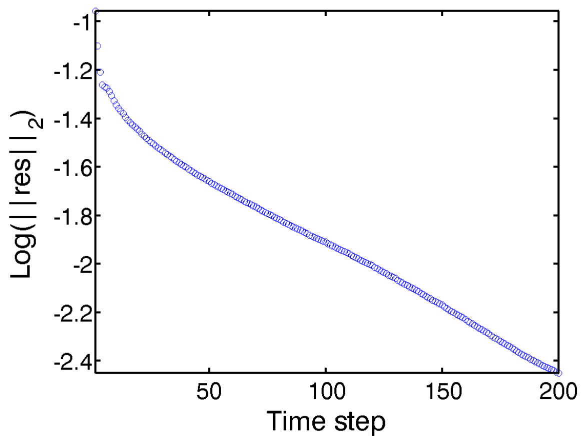


Figure 5.15: Residual history for a damped Newton method for the computation of the steady state around a NACA0012 profile.

The starting CFL number is 4 and this is increased every 10 steps by 1.2. While this is not an optimal strategy, it works reasonably well. The Newton termination tolerance is 10^{-4}, as is the case for the linear systems. As can be seen, convergence to the steady state is very slow, which explains why Newton methods are rarely used for steady state computations.

5.7.3 Computation and Storage of the Jacobian

The computation of the Jacobian leads to two significant problems. First, as explained in the introduction of this chapter, the function $\mathbf{F}$ is in general only piecewise differentiable and thus, the Jacobian in $\underline{\mathbf{u}}^{(k)}$ may not exist. Second, if it exists, the computation of the full second order Jacobian in the finite volume context is extremely difficult to implement and to compute. In the DG context, the implementation and computation is much easier, but in particular the viscous fluxes pose a certain amount of difficulty. Therefore, it is important to consider ways of circumventing these two problems.

First of all, approximations of the Jacobian can be used. For example, we could compute a Jacobian based on the first order method, which does not contain a reconstruction procedure or limiters and can therefore be implemented and computed in a reasonable amount of time. This means that a method of Newton type (5.42) is employed and thus only first order convergence is reached. Nevertheless, this is a popular approach.

Alternatively, we could use automatic differentiation. This is a procedure where the code itself is analyzed by a program (for example ADIFOR [29]), which automatically creates a second version of the code, that computes the derivative using product rule, quotient rule and so on. If-statements are treated as branches of the Jacobian definition.

Both of these approaches have the drawback that the Jacobian needs to be stored. Since it is sparse, this is done using a sparse data format, for example CVS. There, two vectors are stored for each row, one integer vector for the indices of columns where nonzero entries can be found and a second one of equal length with the corresponding values. The block structure can be respected by letting the indices correspond to cells and changing the second vector from one with values to one with pointers, which point to an appropriate container for values of the small block. The number of nonzeros for threedimensional flows is 35 per unknown for quadrilaterals and 25 per unknown for tetrahedrons for finite volume schemes, whereas the number explodes for DG methods, leading to an often prohibitive amount of storage. Therefore, often so called Jacobian free methods will be used in conjunction with Krylov-subspace methods. This will be explained in the next section.

5.8 Krylov subspace methods

As explained in section 5.2, we use Krylov subspace methods for the linear system [169, 204]. These approximate the solution of a linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{A} \in \mathbb{R}^{m\times m}$$

in the space

$$\mathbf{x}_0 + \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, ..., \mathbf{A}^{m-1}\mathbf{r}_0\} = \mathbf{x}_0 + \mathcal{K}_m(\mathbf{A}, \mathbf{b}, \mathbf{x}_0) \tag{5.46}$$

with $\mathbf{x}_0$ being the initial guess. The space $\mathcal{K}_m$ is called the m-th Krylov subspace of $\mathbf{A}, \mathbf{b}$ and $\mathbf{x}_0$. Thus, the approximate solution in step m is of the form $\mathbf{x}_m = \mathbf{x}_0 + p(\mathbf{A})\mathbf{r}_0$ with p being a polynomial of degree $m-1$. A prototype Krylov subspace method computes an orthogonal basis of this space and then uses a projection to compute the next iterate. This process needs the Jacobian only in the form of matrix vector multiplications and thus the sparsity of $\mathbf{A}$ can be exploited.

In the case of symmetric and positive definite systems, the CG method computes the iterate with optimal error in the energy norm $\sqrt{\mathbf{x}^T\mathbf{A}\mathbf{x}}$ using a short recurrence of three vectors. As Faber and Manteuffel proved in [58], a similar method that is both optimal and has a short recurrence is not possible for general matrices. Therefore, a choice has to be made: Either use a method that is optimal and thus has increasing demand for storage and CPU time or use a method that has a short recurrence and thus a constant storage and

CPU time requirement per iteration or use a method like CGN that works with the normal equations and thus the matrix $\mathbf{A}^T\mathbf{A}$. This comes at the price of a squaring of the condition number and the use of $\mathbf{A}^T$, which is why we do not suggest the use of these schemes.

The most prominent methods of the first two kinds are GMRES of Saad and Schultz [170], which computes the optimal iterate regarding the 2-norm of the residual and BiCGSTAB of van der Vorst [203], which has a three-vector recurrence. Furthermore, the interesting method IDR(s) has been recently introduced by van Gijzen and Sonneveld, which is a generalization of BiCGSTAB [182]. In [147], different Krylov subspace methods are applied to different classes of nonsymmetric systems and for each type, a class is found where this class performs best. Therefore, no general answer can be given, as to which method performs best for nonsymmetric matrices. Note that by construction, GMRES is tied to the 2-norm, whereas the others are not.

5.8.1 GMRES and related methods

As a first Krylov subspace method we will explain the generalized minimal residual method (GMRES) [170] in detail. In the j-th iteration, the scheme computes an orthogonal basis $\mathbf{v}_1, ..., \mathbf{v}_j$ of the j-th Krylov subspace

$$\mathcal{K}_j = \text{span}\{\mathbf{r}_0, ..., \mathbf{A}^{j-1}\mathbf{r}_0\}$$

by the so called Arnoldi method. GMRES uses this basis to minimize the functional

$$J(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$$

in the space $\mathbf{x}_0 + \mathcal{K}_j$. The point about Arnoldi's method is that it allows an efficient implementation using a Hessenberg matrix. In every step, this is updated and transformed to an upper triangular matrix using for example Givens-rotations. This then allows to obtain the value of the functional J in every iteration without explicit computation of $\mathbf{x}_j$, which is only done after the tolerance is satisfied. In pseudocode, the algorithm can be formulated as (taken from [141]):

```
• $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
• If $\mathbf{r}_0 = \mathbf{0}$, then END
• $\mathbf{v}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2}$
• For $j = 1, ..., m$
    – $\mathbf{w}_j = \mathbf{A}\mathbf{v}_j$
    – For $i = 1, ..., j$ do $h_{ij} = \mathbf{v}_i^T\mathbf{w}_j$
    – $\mathbf{w}_j = \mathbf{w}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i, \quad h_{j+1,j} = \|\mathbf{w}_j\|_2$
    – For $i = 1, ..., j-1$ do $\begin{pmatrix} h_{ij} \\ h_{i+1,j} \end{pmatrix} = \begin{pmatrix} c_{i+1} & s_{i+1} \\ -s_{i+1} & c_{i+1} \end{pmatrix} \begin{pmatrix} h_{ij} \\ h_{i+1,j} \end{pmatrix}$
    – $\beta = \sqrt{h_{jj}^2 + h_{j+1,j}^2}$;   $s_{j+1} = \frac{h_{j+1,j}}{\beta}$
    – $c_{j+1} = \frac{h_{jj}}{\beta}$;   $h_{jj} = \beta$.
    – $\gamma_{j+1} = -s_{j+1}\gamma_j$;   $\gamma_j = c_{j+1}\gamma_j$
    – if $|\gamma_{j+1}| \geq TOL$, $\mathbf{v}_{j+1} = \frac{\mathbf{w}_j}{h_{j+1,j}}$
    – else
        * for $i = j, ..., 1$ do $\alpha_i = \frac{1}{h_{jj}} \left( \gamma_j - \sum_{k=i+1}^{j} h_{ik}\alpha_k \right)$
        * $\mathbf{x} = \mathbf{x}_0 + \sum_{i=1}^{j} \alpha_i \mathbf{v}_i$
        * END
```

Since GMRES has no short recurrence, the whole basis has to be stored. Thus the cost and storage per step increase linearly with the iteration number. In step j, we need to store the j basis vectors and compute one matrix vector product and j scalar products. It is possible to restart the iteration after k iterations by scrapping the orthogonal basis and starting new with the current approximation, thus bounding the maximal amount of storage without simply terminating the iteration. This method is called GMRES(k). Unfortunately, there are examples where the restart technique does not lead to convergence and examples, where the restart convergence results in a speedup. Typical restart lengths are 30 or 40.

The assumption used in the termination criteria is that the basis is actually orthogonal. This can be tested by computing the residual after the computation of the solution. It turns out that in practice, the difference in the estimate and the actual residual is negligible. Therefore, we will never use techniques for reorthonormalization of the basis used in GMRES, as suggested by several authors.

Due to the minimization, in exact arithmetic, GMRES computes the exact solution of an $m \times m$ linear equation system in at most m steps. Furthermore, the residual in the 2-norm is nonincreasing in every step. For diagonalizable matrices, a more useful residual estimate is possible (for the proof of this and the next results, see e.g. [108]):

Theorem 7 *Let $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ be a nonsingular diagonalizable matrix. Then for all $p \in \Pi_k$ with $p(0) = 1$, we have*

$$\|\mathbf{r}_{n+1}\|_2 \leq \|\mathbf{r}_0\|_2 \kappa_2(\mathbf{V}) \max_{z \in \sigma(A)} |p_k(z)|.$$

For nonnormal matrices, the condition number is not a good tool to describe the behavior of the matrix $\mathbf{A}$. More useful is the distribution of eigenvalues. A result that sheds some light on the performance of GMRES for general matrices is the following:

Theorem 8 *Let* $\mathbf{A}$ *be nonsingular, then we have*

$$\mathbf{r}_k = \min_{p \in \Pi_k, p(0)=1} \|p(\mathbf{A})\mathbf{r}_0\|_2.$$

Using this it can be proved that if $\mathbf{A}$ is normal and the right hand side has only m eigenvector components, GMRES will produce the solution in m iterations. This suggests that the method will converge fast if the eigenvalues are clustered, since then the method will choose the polynomial such that the zeros are the eigenvalues. This insight is useful when designing preconditioners (see the next chapter). However, if a matrix is strongly nonnormal, the eigenvalues fail to accurately reflect the behavior of a matrix. Consequently, a theorem by Greenbaum, Pták and Strakoš [74] states that for any discrete monotone decreasing function, a matrix with an arbitrary eigenvalue distribution can be constructed for which the GMRES algorithm produces that residual history. In particular it might be possible that the residual is constant until the very last step when it drops to zero.

GCR

A method that is mathematically equivalent to GMRES is the Generalised Conjugate Residual method (GCR) [55]. This uses two sets of basis vectors and thus allows some more flexibility in convergence acceleration, which will be discussed later.

```
• $\mathbf{x}_0 = \mathbf{0}$, $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, k=-1
• while $\|\mathbf{r}_k\|_2 > tol$ do
    – $k = k + 1$
    – $\mathbf{p}_k = \mathbf{r}_k$
    – $\mathbf{q}_k = \mathbf{A}\mathbf{p}_k$
    – for $i = 0, 1, \dots, k-1$ do $\alpha_i = \mathbf{q}_i^T\mathbf{q}_i, \mathbf{q}_k = \mathbf{q}_k - \alpha_i\mathbf{q}_i, \mathbf{p}_k = \mathbf{p}_k - \alpha_i\mathbf{p}_i$
    – $\mathbf{q}_k = \mathbf{q}_k/\|\mathbf{q}_k\|_2$, $\mathbf{p}_k = \mathbf{p}_k/\|\mathbf{q}_k\|_2$
    – $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k\mathbf{q}_k^T\mathbf{r}_k$
    – $\mathbf{r}_{k+1} = \mathbf{r}_k - \mathbf{q}_k\mathbf{q}_k^T\mathbf{r}_k$
```

5.8.2 BiCGSTAB

As the major alternative to the GMRES method, the BiCGSTAB method was established [203]. This uses two matrix vector products per iteration, but a constant number of basis vectors and scalar products. The method is constructed to satisfy

$$\mathbf{r}_k = q_k(\mathbf{A})p_k(\mathbf{A})\mathbf{r}_0$$

where

$$q_k(z) = \Pi_{i=1}^k (1 - \omega_i z).$$

In pseudocode, the method can be written as follows:

```
• $\mathbf{r}_0 = \mathbf{b}_0 - \mathbf{A}\mathbf{x}_0$. Set $\rho_0 = \alpha = \omega = 1$
• If $\mathbf{r}_0 = \mathbf{0}$, then END
• For $j = 1, ..., n$
    – $\beta = (\rho_k/\rho_{k-1})(\alpha/\omega)$
    – $\mathbf{p} = \mathbf{r} + \beta(\mathbf{p} - \omega\mathbf{v}$
    – $\mathbf{v} = \mathbf{A}\mathbf{p}$
    – $\alpha = \rho_k/(\hat{\mathbf{r}}_o^T\mathbf{v})$
    – $\mathbf{s} = \mathbf{r} - \alpha\mathbf{v}$, $\mathbf{t} = \mathbf{A}\mathbf{s}$
    – $\omega = \mathbf{t}^t\mathbf{s}/\|\mathbf{t}\|_2^2$, $\rho_{k+1} = -\omega\hat{\mathbf{r}}_0^T\mathbf{t}$
    – $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p} + \omega\mathbf{s}$
    – $\mathbf{r} = \mathbf{s} - \omega\mathbf{t}$
    –       END
```

For BiCGSTAB, no convergence analysis similar to the one of GMRES is known and in fact, the method can break down. What can be proved is that the residual after k steps of BiCGSTAB is never smaller than that after $2k$ steps of GMRES when starting with the same initial guess. Nevertheless, it exhibits fast convergence behavior in practice.

5.9 Jacobian Free Newton-Krylov methods

In Krylov subspace methods, the system matrix appears only in matrix vector products. Thus it is possible to formulate a Jacobian free version of Newton's method. See [113] for a survey on Jacobian free Newton-Krylov methods with a lot of useful references. To this end, the matrix vector products $\mathbf{Aq}$ are replaced by a difference quotient via

$$\mathbf{Aq} = \frac{\partial \underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}})}{\partial \underline{\mathbf{u}}}\underline{\mathbf{q}} \approx \frac{\underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}} + \epsilon\underline{\mathbf{q}}) - \underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}})}{\epsilon}. \tag{5.47}$$

This works for the linear systems arising in Newton scheme, but also for those from the Rosenbrock scheme. For the root equation arising from (5.1), the above approximation translates into

$$\mathbf{A}(\bar{\underline{\mathbf{u}}})\underline{\mathbf{q}} \approx \underline{\mathbf{q}} + \alpha\Delta t\left(\frac{\underline{\mathbf{f}}(\bar{\underline{\mathbf{u}}} + \epsilon\underline{\mathbf{q}}) - \underline{\mathbf{f}}(\bar{\underline{\mathbf{u}}})}{\epsilon}\right).$$

If the parameter ϵ is chosen very small, the approximation becomes better, however, cancellation errors become a major problem. A simple choice for the parameter that avoids cancellation but still is moderately small is given by Quin, Ludlow and Shaw [159] as

$$\epsilon = \frac{\sqrt{eps}}{\|\underline{\mathbf{q}}\|_2},$$

where *eps* is the machine accuracy.

The Jacobian free version has several advantages, the most important are low storage and ease of implementation: instead of computing the Jacobian by analytical formulas or difference quotients, we only need flux evaluations. In the context of finite volume solvers, the most important part of the spatial discretization are the approximate Riemann solvers. Changing one of these thus becomes much more simple. Furthermore, this allows to increase the convergence speed of the Newton scheme: Since we never compute the exact Jacobian, but only a first order version, the scheme with matrix is always a method of Newton-type (5.42) and thus has first order convergence. However, the Jacobian-free approximation is for the second order Jacobian and thus can obtain second order convergence if proper forcing terms are employed, since it is possible to view the errors coming from the finite difference approximation as arising from inexact solves and not from approximation to the Jacobian.

Of the Krylov subspace methods suitable for the solution of unsymmetric linear equation systems, the GMRES method was explained by McHugh and Knoll [138] to perform better than others in the Jacobian free context. The reason for this is that the vectors in matrix vector multiplications in GMRES are normalized, as opposed to those in other methods.

Regarding convergence, there is the following theorem about the outer iteration in the JFNK scheme, when the linear systems are solved using GMRES (see [108]). Essentially, the Jacobian free approximation adds another error of the order $\mathcal{O}(\epsilon)$. This can be interpreted as an increase of the tolerance η_k in the kth step to $\eta_k + c\epsilon$ for some constant c. Then, the previous theorem 6 can be applied:

Theorem 9 *Let the standard assumptions hold. Then there are δ, $\bar{\sigma}$, c such that if $\underline{\mathbf{u}}_0$ is in a δ-neighbodhood of $\underline{\mathbf{u}}^*$ and the sequences $\{\eta_k\}$ and $\{\epsilon_k\}$ satisfy*

$$\sigma_k = \eta_k + c\epsilon_k \leq \bar{\sigma},$$

then the Jacobian free Newton-GMRES iteration (5.43) converges linearly. Moreover,

- *if $\sigma_k \to 0$, the convergence is superlinear and*
- *if $\sigma_k \leq K_\eta \|\mathbf{F}(\underline{\mathbf{u}}^{(k)})\|^p$ for some $K_\eta > 0$ and $p \in [0,1]$, the convergence is superlinear with order $1+p$.*

This theorem says that for superlinear convergence, the parameter ϵ in (5.47) needs to approach zero. However, this leads to cancellation errors. Therefore, if we keep ϵ fixed as described, a behavior like quadratic convergence can only be expected for the initial stages of the iterations, while $c\epsilon$ is still small compared to η_k. The constant c is problem dependent and thus it can happen that linear convergence sets in at a very late stage. In these cases, the strategy of Eisenstat and Walker for the choice of η_k still makes sense.

5.10 Comparison of GMRES and BiCGSTAB

We now compare GMRES, GMRES(25) and BiCGSTAB for the case with matrix and the Jacobian-free variant using TAU_2D. To this end, we consider the linear system from an implicit Euler step with a certain CFL number, solve that up to machine accuracy and look at the residual norm $\|\mathbf{A}x^{(k)} - \mathbf{b}\|_2$ over iterations. The first test problem is the isentropic vortex described in the appendix A.1 and we consider the first time step using the AUSMDV flux function and a first order discretization in space. In figure 5.16 (left), we see that the BiCGSTAB variants converge the fastest, but run into problems at 10^{-10}. GMRES(25) and GMRES have the same convergence behavior, thus restarting has no detrimental effect. However, the Jacobian free variant JF-GMRES(25) starts having problems after the second restart.

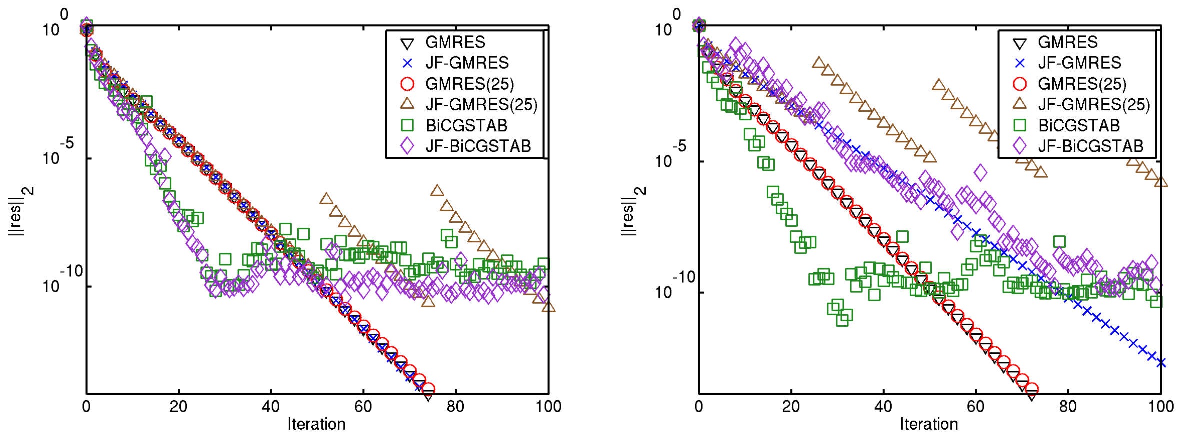


Figure 5.16: Norm of the relative residual over iterations for different Krylov subspace schemes for the Shu-Vortex problem at CFL=5 on the basis of a first order FV scheme (left) and a second order FV scheme (right).

Now, this test is fair in that all methods solve the same system. However, the more realistic situation is a second order discretization in space. Therefore, we repeat the experiment with a linear reconstruction using the Barth limiter. Now, the schemes with a matrix solve a slightly different system with the same matrix as in the first test, but a different right hand side, corresponding to a method of Newton-type. The Jacobian-free schemes approximately solve the system corresponding to the second order Jacobian. As can be seen in figure 5.16 (right), there is very little difference in convergence behavior for the schemes with matrix to the first test case. This is expected, since the system matrix is the same. The Jacobian-free schemes on the other hand converge slower, which can be attributed to the more difficult system. Now, JF-BiCGSTAB performs worse than JF-GMRES.

We now repeat these experiments with CFL 5 and CFL 20 (see figure 5.17) and additionally for the wind turbine problem (described in the appendix A.2) with CFL 5 and 20 (see figure 5.18). It becomes apparent that the larger the CFL number, the more difficult it is for the Krylov subspace methods to solve the system. This corresponds the observations about the system matrix made so far. Actually, for large CFL numbers, JF-BiCGSTAB is not a good method and may even diverge. This is in line with the results of Meister, that for the case with a Jacobian, BiCGSTAB performs best [**?**] and those of McHugh and Knoll that CGS and BiCGSTAB do not perform well in the JFNK setting [138]. Thus, we will not consider BiCGSTAB anymore.

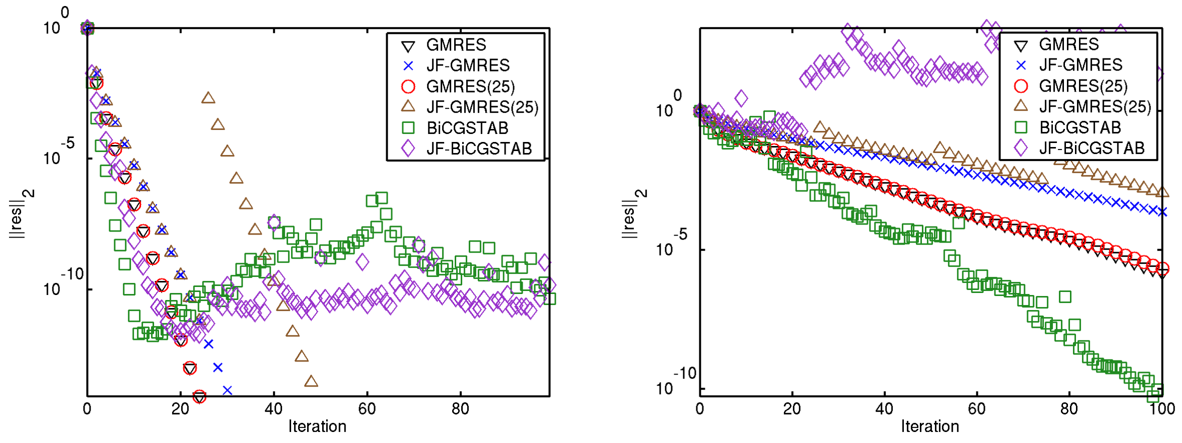


Figure 5.17: Comparison of different Krylov subspace schemes for the Shu Vortex problem at CFL=1 (left) and CFL=20 (right).

5.11 Comparison of variants of Newton's method

To compare the different variants of Newton's method, we use the code TAU_2D with the AUSMDV flux and a linear reconstruction based on the Barth limiter. The nonlinear system considered is the first system appearing when solving the Shu vortex problem A.1 using the

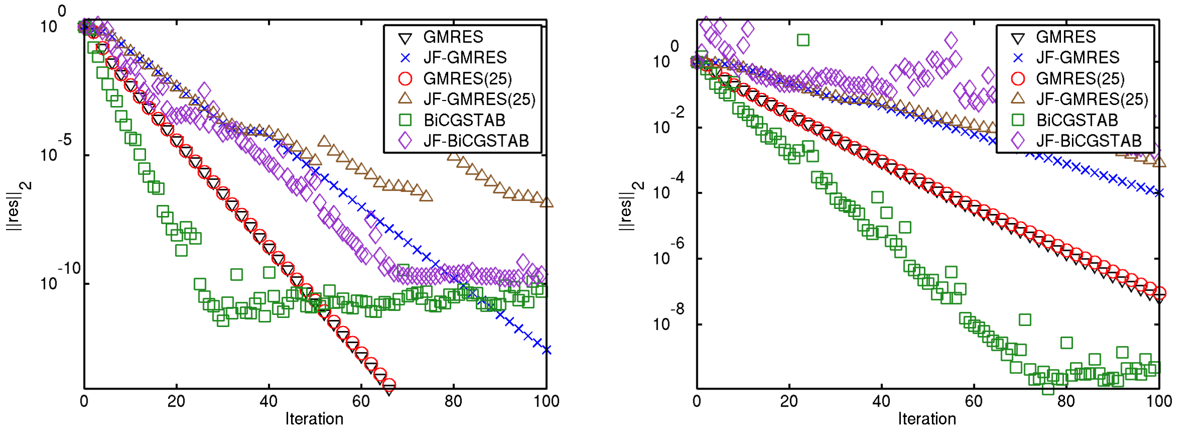


Figure 5.18: Comparison of different Krylov subspace schemes for the Wind turbine problem at CFL=5 (left) and CFL=20 (right).

implicit Euler method with $CFL = 0.7$. The linear systems are solved using GMRES up to machine accuracy. In figure 5.19, the convergence curves for different variants of Newton's method are shown. In the left picture, the actual relative errors are shown, which have been obtained by solving the nonlinear system up to machine accuracy, storing the solution and repeating the Newton loop. The figure on the right shows the relative residuals. As can be seen, the method with a first order Jacobian is first order convergent, as expected. If the Eisenstat-Walker strategy is used, this hardly changes errors or residuals and thus leads to the same order of convergence. The JFNK method exhibits second order convergence and the Eisenstat-Walker strategy is able to obtain that as well, where again, errors and residuals are hardly changed. This demonstrates that the strategy used is indeed very reasonable.

Furthermore, for this problem, the JFNK scheme obtains quadratic convergence up to machine accuracy, meaning that the constant c from theorem (9) is very small. The other thing that can be seen is that for this case, the residual is an excellent indicator of the error, being larger by less than a factor of five.

Now, we estimate the radius of convergence by using the canonical initial guess $\mathbf{u}^{(0)} = \mathbf{u}_n$ and determine the minimal time step Δt at which the Newton iteration diverges. Intuitively, one assumes that when we increase Δt, $\mathbf{u}_{n+1}$ gets further away from $\mathbf{u}_n$. However, this need not be the case. For example for a periodic flow, it might be that an increase in Δt causes $\mathbf{u}_{n+1}$ to get closer to $\mathbf{u}_n$. Therefore, we choose as test case the Shu vortex where we know that the solution essentially moves to the right with time. The results can be seen in table 5.8. As can be seen, the methods of Newton type with large errors in the Jacobian lead to a smaller radius of convergence, whereas the JFNK methods have a radius of convergence that is about four times larger. The use of the Eisenstat-Walker strategy as opposed to a

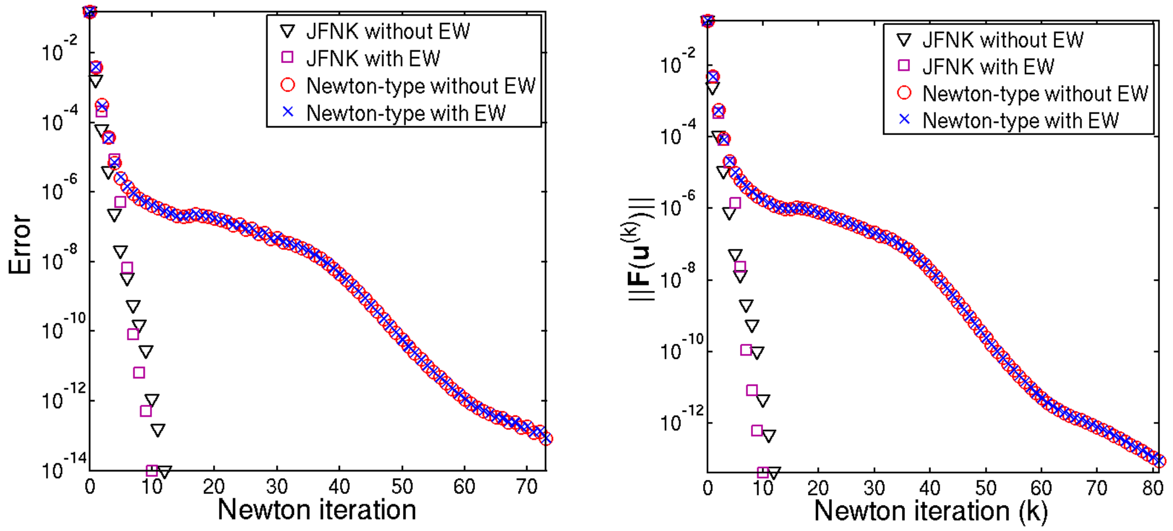


Figure 5.19: Illustration of Newton's method. Relative errors (left) and relative residuals (right).

JFNK-FT	JFNK-EW	Newton-type-FT	Newton-type-EW
3.0	4.1	0.7	0.9

Table 5.8: Upper bounds of convergence radius for Shu vortex problem in terms of CFL numbers. FT stands for a fixed tolerance, EW for Eisenstat-Walker.

fixed tolerance also leads to an increase in the convergence radius.

We now compare the efficiency of the different schemes. To this end, we solve the wind turbine problem A.2 on a time interval of ten seconds using time adaptive SDIRK2 with a relative and absolute tolerance of 10^{-2}. The nonlinear systems are solved with up to 40 Newton steps and the linear systems are solved using GMRES. Regarding the other tolerances, the Newton tolerance is set to $10^{-2}/5$, whereas the linear tolerance is set to $10^{-2}/50$ for the JFNK case and to 10^{-2} for the Newton type case. This is because in the latter, we only expect first order convergence of the Newton method and thus, it is not necessary to solve the linear systems to such a high accuracy.

In table 5.9, the total number of GMRES iterations and the total CPU time are shown. The computations are performed on one CPU of an Opteron Quad twelve-core 6168 machine with 1.9 GHz. The computation for the method of Newton-type with a fixed tolerance was stopped after more than 50.000 time steps needing more than 4 Million GMRES iterations and still not having computed more than 0.6 seconds of real time. With the Eisenstat-

	JFNK-FT	JFNK-EW	Newton-type-FT	Newton-type-EW
Iter.	379,392	348,926	-	1,121,545
CPU in s	64,069	61,337	-	348,174

Table 5.9: Comparison of efficiency of different Newton variants NEW GRID!.

Walker strategy, the computations take more than five times longer than with the JFNK methods. So in both cases, the use of the Eisenstat-Walker strategy leads to a speedup, although it is less pronounced for the JFNK methods. Thus, the JFNK method is faster than the method of Newton-type, meaning that the fastest method is the JFNK method with the Eisenstat-Walker strategy. This can be attributed to a number of factors playing together.

First, as just demonstrated, the JFNK method has a larger convergence radius than the method of Newton type. This means that the control that causes the time step to be repeated with a smaller time step when Newton's method fails the tolerance test after the maximal number of iterations, kicks in less often and actually leads to larger possible time steps. Second, the JFNK method is second order convergent, needing less Newton steps, thus being more efficient and having again the same effect as the first issue. Third, the Eisenstat Walker strategy reduces the tolerance at which the linear systems are solved. This is good in itself, since it leads to less Krylov subspace iterations, but there is an added benefit when using GMRES: Since GMRES needs more storage and more computational effort with every iteration, it is extremely fast for small tolerances. Finally, we have seen in the last section that GMRES in the JFNK context works better if we avoid restarting. This is achieved if the tolerances are such that GMRES terminates before the maximal dimension of the Krylov subspace is reached.

Chapter 6

Preconditioning linear systems

As discussed in the last chapter, the speed of convergence of Krylov subspace methods depends strongly on the matrix. Unfortunately, matrices arising from the discretization of PDEs are typically such that Krylov subspace methods converge slowly. Therefore, an idea called preconditioning is used to transform the linear equation system into an equivalent one to speed up convergence:

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

Here, $\mathbf{P}_L$ and $\mathbf{P}_R$ are invertible matrices, called a left respectively right preconditioner that approximate the system matrix in a cheap way. The optimal preconditioner in terms of accuracy is $\mathbf{A}^{-1}$, since then we would just need to solve a system with the identity matrix. However, that's not efficient. In terms of cost, the optimal preconditioner is the identity, which comes for free, but also has no effect on the number of iterations.

Often, the preconditioner is not given directly, but implicitly via its inverse. Then, the application of the preconditioner corresponds to the solution of a linear equation system. If chosen well, the speedup of the Krylov subspace method is enormous and therefore, the choice of the preconditioner is more important than the specific Krylov subspace method used.

Preconditioning can be done very easily in Krylov subspace methods. Every time a matrix vector product $\mathbf{A}\mathbf{v_j}$ appears in the original algorithm, the right preconditioned method is obtained by applying the preconditioner $\mathbf{P}_R$ to the vector $\mathbf{v}_j$ in advance and then computing the matrix vector product with $\mathbf{A}$. For left preconditioning the preconditioner is applied afterwards instead. Hence, left preconditioned GMRES works in the Krylov subspace

$$\mathcal{K}_k(\mathbf{PA}, \mathbf{r}_0^P) = \text{span}\{\mathbf{r}_0^P, \mathbf{PA}\mathbf{r}_0^P, ..., (\mathbf{PA})^{k-1}\mathbf{r}_0^P\},$$

whereas the Krylov subspace generated by right preconditioning is

$$\mathcal{K}_k(\mathbf{AP}, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, \mathbf{AP}\mathbf{r}_0, ..., (\mathbf{AP})^{k-1}\mathbf{r}_0\}.$$

Note that right preconditioning does not change the initial residual, because

$$\mathbf{r}_0 = \mathbf{b}_0 - \mathbf{A}\mathbf{x}_0 = \mathbf{b}_0 - \mathbf{A}\mathbf{P}_R\mathbf{x}_0^P,$$

therefore the computation of the initial residual can be done without the right preconditioner. However, when the tolerance criterion is fulfilled, the right preconditioner has to be applied one last time to change back from the preconditioned approximation to the unpreconditioned. On the other hand, a left preconditioner has to be applied once initially, but not afterwards. This means that left preconditioning changes the residual.

For nonnormal matrices as we have here, the crucial properties that determine the speed of convergence are the pseudospectra that were introduced by Trefethen [199]. Unfortunately, for the matrix dimensions considered here, these cannot be computed in reasonable time. Furthermore, there exist only a few analytical results about the matrices appearing in compressible flow. Therefore, the preconditioner has to be chosen by numerical experiments and heuristics. An overview of preconditioners with special emphasis on application in flow problems can be found in the book of Meister [141] and the study in the context of compressible flow by Meister and Vömel [143].

6.1 Preconditioning for JFNK schemes

Regarding preconditioned versions of the Jacobian free matrix vector product approximation (5.47), we obtain for left preconditioning

$$\mathbf{P}\mathbf{A}\mathbf{q} \approx \mathbf{P}\left(\frac{\underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}} + \epsilon\underline{\mathbf{q}}) - \underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}})}{\epsilon}\right) \tag{6.1}$$

whereas for right preconditioning we have

$$\mathbf{A}\mathbf{P}\mathbf{q} \approx \frac{\underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}} + \epsilon\mathbf{P}\underline{\mathbf{q}}) - \underline{\mathbf{F}}(\bar{\underline{\mathbf{u}}})}{\epsilon}. \tag{6.2}$$

Thus, preconditioning can be implemented in exactly the same way as for the case with Jacobian. However, the construction of the preconditioner becomes a problem, since we do not have a Jacobian anymore. In fact, while the JFNK scheme as presented so far can be implemented into an existing code with little extra work and in particular, no new data structures, it is here that things can become complicated. Regarding this, there are two approaches:

1. Compute the Jacobian nevertheless, then compute the preconditioner and store that in place of the Jacobian

2. Use preconditioners that need only parts of or no part of the Jacobian, thus leading to extremely low storage methods.

The first strategy still reduces the storage needed by one matrix compared to the case with Jacobian. It is for example necessary when using ILU preconditioning. The second strategy severely limits the set of possible preconditioners. Jacobi would be one example, since this just uses the diagonal. Another example would be multigrid methods with appropriate smoothers, in particular Jacobi or Runge-Kutta smoothing.

6.2 Specific preconditioners

6.2.1 Block preconditioners

As discussed in section 5.2, the matrix we are concerned with is a block matrix and if this structure is respected by the preconditioner, the convergence speed is increased. This is done by interpreting all methods if possible as block methods with an appropriately chosen block size. The main difference of preconditioners for finite volume methods as opposed to discontinuous Galerkin methods is the size of the blocks, which is significantly larger for DG schemes. This makes an efficient treatment of the blocks imperative for a successful implicit DG scheme, whereas it is only beneficial for a finite volume scheme.

In the case of the DG-SEM method, there are different block sizes that can be used (compare figure 5.1). First of all, there are the small blocks corresponding to one degree of freedom which are of the same size as the number of equations, namely $d + 2$. Then, this can be increased to include all degrees of freedom in x_1-direction, leading to a block of size $(d + 2)(p + 1)$. Of course, the x_2 direction can be included as well, until if also the x_3 direction is included the block corresponds to all the unknowns in one cell. For the modal-nodal DG scheme, we later suggest the ROBO-SGS method, which exploits the specific block structure of the hierarchical basis.

6.2.2 Splitting-methods

The splitting methods explained in the last chapter, although inferior to Krylov subspace methods for the solution of linear systems, turn out to provide useful preconditioners. They were based on the splitting

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

giving rise to the fixed point method (5.12):

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{B}^{-1}\mathbf{A})\mathbf{x}^{(k)} + \mathbf{B}^{-1}\mathbf{b}.$$

Thus, they are based on approximating $\mathbf{A}^{-1}$ via the matrix $\mathbf{B}$. Therefore, $\mathbf{B}$ can be used as a preconditioner. In particular the symmetric block Gauss-Seidel-method (SGS) is a very good preconditioner for compressible flow problems. One application of SGS as a preconditioner to the vector $\mathbf{q}$ corresponds to solving the equation system (see (5.14))

$$(\mathbf{D}+\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{U})\mathbf{x}=\mathbf{q}. \tag{6.3}$$

For the purpose of preconditioning, one iteration is completely sufficient. The blocks of **L** and **U** can be computed when required one at a time, so it is not necessary to store the complete matrix. Only the diagonal blocks which appear several times are computed in advance and stored.

Another simple choice for a splitting method would be Jacobi-preconditioning, corresponding to $\mathbf{P}=\mathbf{D}^{-1}$ (see 5.15), where we need only the diagonal blocks, thus reducing the cost of applying the preconditioner and a preconditioner that is simple to implement. However this rarely pays, since the approximation of **A** is just not good enough.

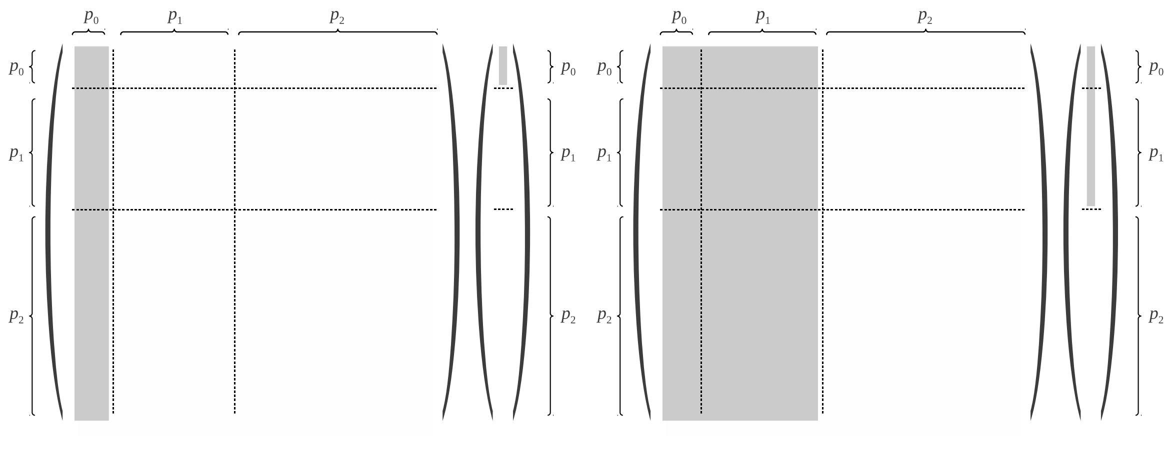
A common technique to improve the efficiency of a splitting related preconditioner is relaxation, which corresponds for SGS to:

$$\frac{1}{\omega(2-\omega)}(\mathbf{D}+\omega\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\omega\mathbf{U})\mathbf{x}=\mathbf{q}.$$

Unfortunately, it is difficult to find a choice of ω that leads to consistently better results and significant speedup can not be expected.

6.2.3 ROBO-SGS

In [23], the low-memory SGS preconditioner with a reduced offblock order (ROBO-SGS) is suggested for use with modal DG schemes. This exploits that the basis is hierarchical in that for the off-diagonal blocks **L** and **U** in (6.3), only the parts corresponding to a lower order discretization are computed, see figure 6.1. The number of entries of the off-diagonal



(a) $p=0$ variant (b) $p=1$ variant

Figure 6.1: Reduced versions of the off-block Jacobians, $p=0$ and $p=1$ variants

blocks in the preconditioner is then $N\cdot(d+2)\times\hat{N}\cdot(d+2)$, where $\hat{N}$ is the dimension of the lower order polynomial space of order $\hat{p}$ chosen for the off-diagonal blocks. We denote the corresponding preconditioner as ROBO-SGS-$\hat{p}$, for example ROBO-SGS-1 is the SGS preconditioner where in the off diagonal blocks, only the terms of order one and zero are considered. Thus, the unknowns whose contributions are incorporated into the preconditioner have a physical meaning even for low orders.

While this results in a decreased accuracy of the preconditioner, the memory requirements and the computational cost of the application become better, the less degrees of freedom of the neighboring cells one takes into account. We consider the significant savings in memory to be even more important for 3D simulations since there memory is usually the main limitation for high order DG schemes.

6.2.4 ILU preconditioning

Another important class of preconditioners are block incomplete LU (ILU) decompositions, where the blocks correspond to the small units the Jacobian consists of. A thorough description can be found in the book of Saad [169]. The computation of a complete LU decomposition is quite expensive and in general leads to a dense decomposition, also for sparse matrices. By prescribing a sparsity pattern, incomplete LU decompositions can be defined, which consist of an approximate factorization $\mathbf{LU} \approx \mathbf{A}$. The larger the sparsity pattern, the better the approximation and the more powerful the preconditioner, but the more expensive are application and computation. The application of such a decomposition as a preconditioner is done by solving by forward-backward substition the corresponding linear equation system.

Given a sparsity pattern $M \subset \{(i,j)|i \neq j, 1 \leq i,j \leq m\}$, the algorithm can be written as

```
• For $k = 1, ..., m-1$
•     for $i = k+1, ..., m$ and if $(i,k) \notin M$
•         $a_{ik} = a_{ik}/a_{kk}$
•         for $j = k+1, ..., m$ and if $(i,j) \notin M$
•             $a_{ij} = a_{ij} - a_{ik}a_{kj}$
•         end for
•     end for
• end for
```

The question remains, how the sparsity pattern should be chosen. A widely used strategy is the level of fill l, leading to the preconditioner ILU(l). The level of fill is a recursively defined measure for how much beyond the original sparsity pattern is allowed: Using the sparsity pattern of $\mathbf{A}$ corresponds to level 0. The sparsity pattern for level 1 is obtained by computing ILU(0), multiplying the corresponding $\mathbf{L}$ and $\mathbf{U}$ matrices and using the nonzero entries of that product. Level 2 is obtained by computing ILU(1), multiplying the corresponding matrices and so on. For the purpose of memory allocation, these patterns can be determined in advance based on the sparsity pattern of A. Those decompositions with higher levels of fill are very good black box preconditioners for flow problems, see for example the study [18]. However they also have large memory requirements. Thus remains ILU(0), which has no additional level of fill beyond the sparsity pattern of the original matrix $\mathbf{A}$.

Another widely used method is ILUT, where the T stands for drop tolerance. There, the sparsity pattern is determined on the fly, by dropping an entry in the $\mathbf{LU}$ factorization, if it is below a drop tolerance ϵ. This leads to a preconditioner with a good compromise between accuracy and efficiency. However, the implementation is much more difficult, due to the variable sparsity pattern.

Note that as GS and SGS, the ILU depends on the ordering of the unknowns, as does its performance. This is because the ordering of unknowns changes the amount of fill-in of the exact LU decomposition and thus also the errors introduced by the ILU. Strategies to do this are for example reverse Cuthill-McKee ordering [169], as well as the physical reordering suggested for SGS in [143]. There, unknowns are renumbered along planes normal to the flow direction, thus the numeration partly reflects the physical flow of information.

6.2.5 Multilevel preconditioners

Another possibility is to use linear multilevel schemes as preconditioners. Using nonlinear schemes as preconditioners is covered in the next subsection. First of all, the linear multigrid methods as described in section 5.6 can be applied. Then, the multi-p-method can be applied to the linear problem. Finally, there are other generalizations of multigrid methods to multilevel ideas, for example algebraic multigrid. All of these can be applied in the Jacobian-free context by replacing the matrix vector products in the algorithm with the finite difference approximation and furthermore, if the coarse grid problem is defined by the coarse grid discretization, instead of a somehow transformed fine grid matrix.

In the finite volume context, this has been analyzed in [136] and found to be competitive when compared to FAS multigrid or using Newton with Multigrid as a linear solver. A number of approaches have been tried in the context of DG methods, e.g. multigrid methods and multi-p methods with different smoothers (see [148, 137] for steady Euler flows), as well as a variant by Persson and Peraire [157], where a two-level multi-p method is used with ILU(0) as a postsmoother.

More precise, they suggest a slightly different method than the linear multigrid described earlier:

1. Restrict right hand side $\mathbf{b}$

2. Solve coarse scale problem for approximation of solution

3. Prolongate solution
4. Apply smoother to residual and correct solution

We propose to use the more classic variant with presmoothing, which can be written as

1. Apply presmoothing
2. Restrict residual $\mathbf{b} - \mathbf{A}\mathbf{x}$
3. Solve coarse scale problem for approximation of error
4. Prolongate error
5. Correct solution

6.2.6 Nonlinear preconditioners

If there is an already existing nonlinear multigrid code with dual time stepping for unsteady flows, it seems natural to use that method as a preconditioner in a JFNK method (5.47), since this gives a low storage preconditioner at no additional implementation cost. This approach was first tried for steady problems by Wigton, Yu and Young in 1985 [223], later by Mavriplis [136] and then for unsteady problems by Bijl and Carpenter [16]. In [26] and [25], Birken and Jameson analyze the use of general nonlinear solvers as preconditioners. This section follows these two articles. It turns out that the use of nonlinear left preconditioning alters the equation system in a nonequivalent way, leading to a stall in Newton convergence.

Left Preconditioning

Following Mavriplis, we define the nonlinear preconditioner for the Jacobian-free method via

$$-\mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}}) := \mathbf{N}(\underline{\mathbf{u}}) - \underline{\mathbf{u}}. \tag{6.4}$$

Here, $\mathbf{N}(\underline{\mathbf{u}})$ is some nonlinear method for the solution of the original nonlinear equation $\mathbf{F}(\underline{\mathbf{u}}^{n+1}) = \mathbf{0}$, for example FAS multigrid. The definition is motivated by noticing that $-\mathbf{F}(\underline{\mathbf{u}})$ is the right hand side of the linear system in a Newton iteration at iterate $\underline{\mathbf{u}}$ (compare (5.40)). Thus, $-\mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}})$ should be defined such that it approximates $\Delta\underline{\mathbf{u}}$. Now, $\underline{\mathbf{u}} + \Delta\underline{\mathbf{u}}$ is supposedly a better approximation of $\underline{\mathbf{u}}^{n+1}$ than $\underline{\mathbf{u}}$, as is $\mathbf{N}(\underline{\mathbf{u}})$, therefore $\mathbf{N}(\underline{\mathbf{u}}) - \underline{\mathbf{u}}$ is a reasonable approximation of $-\mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}})$.

Since $\mathbf{N}$ is nonlinear, we expect $\mathbf{P}^{-1}$ to be changing with every step, so the space in which the Krylov subspace method works would be

$$\mathbf{x}_0 + \text{span}\{\mathbf{P}_0^{-1}\mathbf{r}_0, \mathbf{P}_1^{-1}\mathbf{A}\mathbf{P}_0\mathbf{r}_0, \mathbf{P}_2^{-1}\mathbf{A}\mathbf{P}_1^{-1}\mathbf{A}\mathbf{P}_0^{-1}\mathbf{r}_0, ...\}.$$

This is in general not a Krylov subspace. However, for the Jacobian-free method we have the following result:

Lemma 1 *In the Jacobian-free sense, the left preconditioned operator is linear and is given by*

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{q} = \left(\mathbf{I} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}}\right)\Bigg|_{\underline{\mathbf{u}}^{(k)}} \mathbf{q}. \tag{6.5}$$

The preconditioner is given by

$$\mathbf{P}^{-1} = \left(\mathbf{I} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}}\right)\Bigg|_{\underline{\mathbf{u}}^{(k)}} \mathbf{A}^{-1}$$

Proof: We have for the preconditioned Jacobian-free matrix vector product

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{q} = \frac{\mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}}^{(k)} + \epsilon\mathbf{q}) - \mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}}^{(k)})}{\epsilon}.$$

Inserting (6.4) into the above equation, we obtain

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{q} = \frac{-\mathbf{N}(\underline{\mathbf{u}}^{(k)} + \epsilon\mathbf{q}) + \underline{\mathbf{u}}^{(k)} + \epsilon\mathbf{q} + \mathbf{N}(\underline{\mathbf{u}}^{(k)}) - \underline{\mathbf{u}}^{(k)}}{\epsilon} = \mathbf{q} - \frac{\mathbf{N}(\underline{\mathbf{u}}^{(k)} + \epsilon\mathbf{q}) - \mathbf{N}(\underline{\mathbf{u}}^{(k)})}{\epsilon}.$$

In the Jacobian-free sense, this is nothing but (6.5). The representation of the preconditioner is obtained by reformulating the equation. □

Thus, while the preconditioner is nonlinear, it behaves as if it were a linear operator and may be applied to any Krylov subspace method without changes. We will now use the formulation (6.5) to look more closely at the properties of the preconditioner. In particular, it becomes clear that the preconditioned operator $\mathbf{I} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}}|_{\underline{\mathbf{u}}^{(k)}}$ is not necessarily better than $\mathbf{A}$ as far as convergence is concerned. For the special case of the dual time stepping method, the preconditioner is equal to the original value plus an update from the multigrid method: $\mathbf{N}(\underline{\mathbf{u}}) = \underline{\mathbf{u}} + \mathbf{MG}(\underline{\mathbf{u}})$. We thus obtain

$$\mathbf{I} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}} = \frac{\partial \mathbf{MG}}{\partial \underline{\mathbf{u}}}.$$

If the dual time stepping stalls, for example because we are near a steady state, this is close to zero and may be ill conditioned and thus hinder convergence.

More importantly, the problem is that the preconditioned right hand side is off. In the current method, the definition of the preconditioner is applied when computing the preconditioned right hand side:

$$-\mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}}^{(k)}) = \mathbf{N}(\underline{\mathbf{u}}^{(k)}) - \underline{\mathbf{u}}^{(k)}.$$

But, as we just saw, the correct approach would be to apply (6.5), resulting in

$$-\left(\mathbf{I} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}}\right)\mathbf{A}^{-1}\mathbf{F}(\underline{\mathbf{u}}^{(k)}) = \left(\mathbf{I} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}}\right)\mathbf{\Delta u}^{(k)} = \underline{\mathbf{u}}^{(k+1)} - \underline{\mathbf{u}}^{(k)} - \frac{\partial \mathbf{N}}{\partial \underline{\mathbf{u}}}\mathbf{\Delta u}^{(k)}. \tag{6.6}$$

We thus obtain the following theorem:

Theorem 10 *The nonlinear left preconditioned method changes the right hand side in the Newton method in a nonequivalent way.*

This theorem says that even if the nonlinear left preconditioner increases the convergence speed of the Krylov subspace method, it changes the behavior of Newton's method. Since it is the right hand side that determines the limit of the Newton iteration, we cannot expect fast convergence of the nonlinear iteration anymore. In fact, the numerical evidence shows that it leads to a stall in convergence of the nonlinear iteration. This is demonstrated in figure 6.2 (left), where the convergence curve during one time step for steady inviscid flow around a NACA0012 profile at Mach 0.796 is shown. The grid was a 192×32 C-mesh.

In figure 6.2 (right), this is shown for one system arising from the simulation of a two dimensional flow around a cylinder, just before the onset of turbulence. To be more precise, the code used was a modification of UFLO 103 of Jameson, the grid was a 512×64 C-type grid, the Reynolds number was 100.000 and the Mach number 0.25. As can be seen, the left preconditioned method leads to a stall of Newton convergence, whereas the unpreconditioned method results in fast convergence.

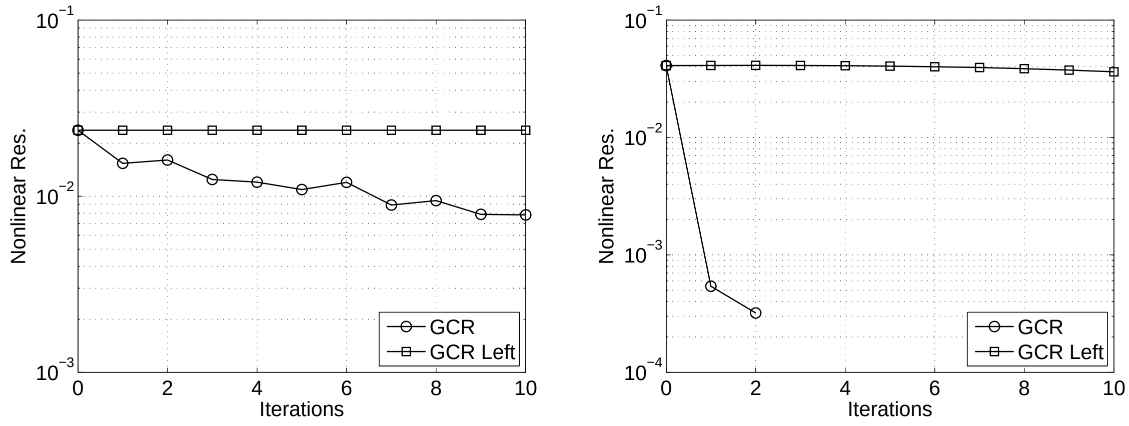


Figure 6.2: Nonlinear residual over Newton iterations for steady inviscid flow (left) and unsteady viscous flow around a cylinder (right). The change of the right hand side by the nonlinear left preconditioning leads to a stall of the convergence of Newton's method.

Note that this cannot be fixed easily since $\underline{\mathbf{u}}^{(k+1)}$ is an unknown. One approach would now be to approximate the right hand side of (6.6), but the most reasonable approximation is $\underline{\mathbf{u}}^{(k)}$ and then we would end up with a zero right hand side and no update for the Newton iteration.

Right Preconditioning

Alternatively we can use the nonlinear preconditioner in the context of right preconditioning and obtain

$$\mathbf{A}\mathbf{P}^{-1}\mathbf{q} \approx \frac{\mathbf{F}(\underline{\mathbf{u}}^{(k)} + \epsilon \mathbf{P}^{-1}\mathbf{q}) - \mathbf{F}(\underline{\mathbf{u}}^{(k)})}{\epsilon},$$

which means that before applying $\mathbf{A}$, we have to apply the preconditioner to $\mathbf{q}$. In the nonlinear case the following problems occur:

1. GMRES uses basisvectors of the solution space, thus these vectors correspond to $\Delta\underline{\mathbf{u}}$. However, the FAS multigrid does not work on differences of solutions, but on the solution vector $\underline{\mathbf{u}}$, therefore, it cannot be used.

2. Since $\mathbf{P}^{-1}$ might be variable, we do not really know what the proper backtransformation would be to transform the preconditioned solution back to the unpreconditioned space.

The second problem is solved by the flexible GMRES method (FGMRES) of Saad [168], which allows to use a different preconditioner in every iteration. However, since FMGRES is just a slight modification of GMRES, it still has the first problem. Both problems are solved by GMRES-* [205] of van der Vorst and Vuik, which allows nonlinear right preconditioning, there the * represents the right preconditioner. Thus, we could have GMRES-SGS or GMRES-DTS (for Dual Time Stepping). GMRES-* is mathematically equivalent to GCR, which is in the fixed preconditioner case mathematically equivalent to GMRES. The preconditioner in GMRES-* is applied by replacing the line $\mathbf{p}_k = \mathbf{r}_k$ in the GCR algorithm with the application of the preconditioner to the residual $\mathbf{r}_k$ and the storing of the result in the search direction $\mathbf{p}_k$. Thus, the preconditioner works with residual vectors and nonlinear right preconditioning is applied via:

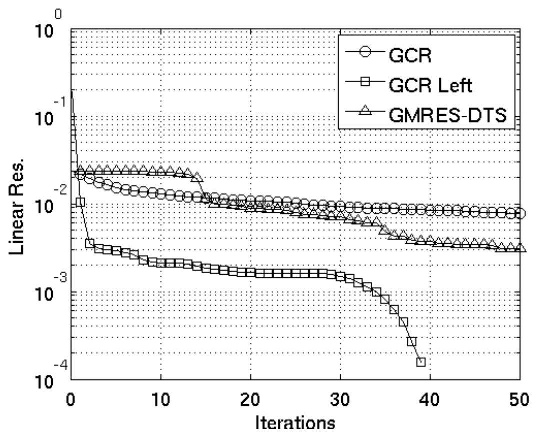
$$\mathbf{P}^{-1}\mathbf{r}_m \approx \mathbf{P}^{-1}\mathbf{F}(\underline{\mathbf{u}}^{(k)} + \mathbf{x}_m) = \underline{\mathbf{u}}^{(k)} + \mathbf{x}_m - \mathbf{N}(\underline{\mathbf{u}}^{(k)} + \mathbf{x}_m), \tag{6.7}$$

which is motivated by remembering that

$$\mathbf{r}_m = \mathbf{A}\mathbf{x}_m - \mathbf{b} = \mathbf{F}(\underline{\mathbf{u}}^{(k)}) + \left.\frac{\partial \mathbf{F}}{\partial \underline{\mathbf{u}}}\right|_{\underline{\mathbf{u}}^{(k)}} (\underline{\mathbf{u}}^{(k+1)} - \underline{\mathbf{u}}^{(k)}) \approx \mathbf{F}(\underline{\mathbf{u}}^{(k)} + \mathbf{x}_m).$$

This is a truly nonlinear method, which does not have the same problem as the left preconditioner of changing the right hand side of the Newton scheme.

The residual history of left and right preconditioning for different test cases is shown in figure 6.2.6. The same test cases as above were used, where the system solved is that from the first Newton iteration. As can be seen, the left preconditioner improves the convergence speed in the unsteady case, but not in the steady case. There, $\mathbf{N}(\underline{\mathbf{u}})$ is close to the identity. The right preconditioner on the other hand, which does not lead to a stall in the Newton iteration, also does not improve the convergence speed much. In the case of steady flow, it even leads to a loss of speed of convergence. We conclude that this type of nonlinear preconditioning is not advisable.



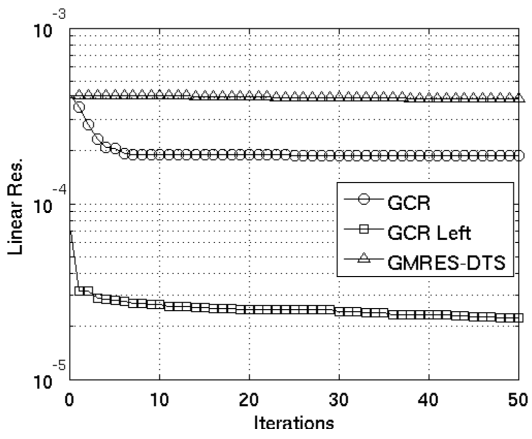


Figure 6.3: Linear Residual versus Iteration number for one linear system of unsteady viscous flow (left) and steady inviscid flow (right).

6.2.7 Other preconditioners

There is a multitude of other preconditioners that have been suggested over time. One example are polynomial preconditioners. There, it is used that if $\rho(\mathbf{I}-\mathbf{A}) < 1$, the inverse of $\mathbf{A}$ can be represented by its Neumann series

$$\mathbf{A}^{-1} = \sum_{k=0}^{\infty} (\mathbf{I}-\mathbf{A})^k.$$

Thus,

$$\mathbf{P} = \sum_{k=0}^{m} (\mathbf{I}-\mathbf{A})^k \tag{6.8}$$

or more generally

$$\mathbf{P} = \sum_{k=0}^{m} \Theta_k (\mathbf{I}-\mathbf{A})^k \tag{6.9}$$

with $\Theta_k \in \mathbb{R}$ can be used as a preconditioner.

Furthermore, there are physical preconditioners that use the discretization of simpler problems that still incorporate core issues of stiffness like low Mach numbers, diffusion or grid induced stiffness to define a preconditioner [154]. Here, it is important to include the sources of stiffness, as for example using the Euler equations to define a preconditioner does not lead to a fast method. This is because in this case, the simplified model is less stiff than the approximated model, the Navier-Stokes equations.

Other ideas would be the use of the Alternate Direction Implicit (ADI) splitting for structured grids [3] or the SPAI preconditioner aimed at parallel computations [76].

6.2.8 Comparison of preconditioners

We will now compare the effect of different preconditioners on the convergence behavior of Krylov subspace methods for a single system. Since this does not take into account the performance of the overall solution scheme for an unsteady flow problem, we cannot say what the best preconditioner is. Appropriate results will be discussed in the next chapter.

We first analyze the influence of ordering on convergence. To this end, we consider the flow around a cylinder at Mach 10. We solve this using the in house FV code TAU_2D on a grid with 20994 point, thus we have 83976 unknowns. We solve the first system when using the implicit Euler method with $CFL = 2.0$ with ILU preconditioned Jacobian-free GMRES and GMRES. As can be seen in figure 6.4, the physical renumbering leads to a small, but noticable increase in convergence speed for the case with a Jacobian, but doesn't change much for the Jacobian-free case, where we have to remember that the linear system solved in that case is the second order system, being more difficult. For other cases with smaller Mach numbers, the physical renumbering strategy has no effect.

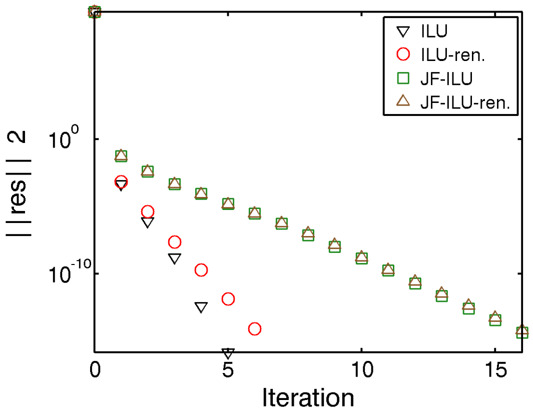


Figure 6.4: Effect of physical renumbering.

To evaluate the effect of using different preconditioners for finite volume schemes, we considere the wind turbine problem A.2 and solve it using the in house FV code TAU_2D. We solve the first system when using the implicit Euler method. In figure 6.5, we can see that Jacobi preconditioning has essentially no effect on the convergence speed, in fact it is barely able to lead to a speedup at all. By contrast, ILU preconditioning increases convergence speed by more than a factor of two. As before, for the more difficult system with $CFL = 20$, even the ILU preconditioned scheme needs a significant number of iterations. However, note that for engineering accuracies of 10^{-3} or 10^{-5}, ILU is able to prevent a restart.

For the case of DG methods in parallel, we compare Jacobi, ILU, ILU-CSC and ROBO-SGS-0 till ROBO-SGS-4 for a 5th order polymorphic modal-nodal discretization of the sphere vortex shedding problem A.3. We use eight CPUs in a domain decomposition

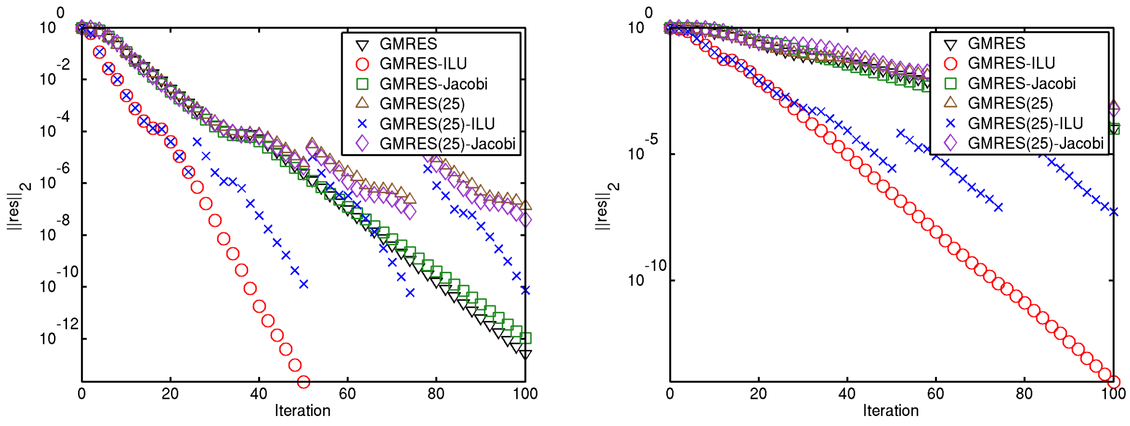


Figure 6.5: Comparison of different preconditioners for FV. CFL=5 (left) and CFL=20 (right).

fashion, as explained in section 7.3. The system solved is the first system arising in the Newton scheme for the first nonlinear system during the first time step of an ESDIRK4 time integration scheme with a time step size of $\Delta t = 0.065$. As a solver, Jacobian-Free GMRES(20) is employed. The relative residuals are shown in figure 6.6.

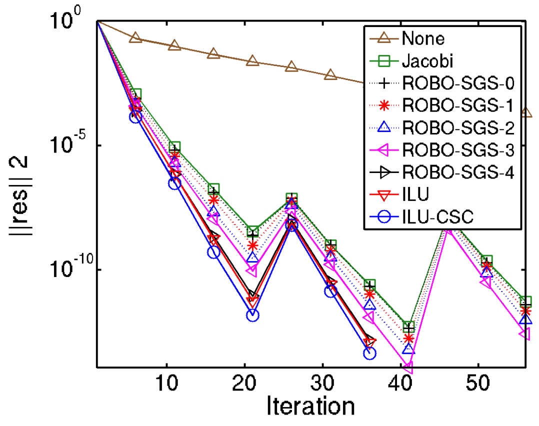


Figure 6.6: Comparison of different preconditioners for DG discretization.

As can be seen, the most powerful preconditioner is ILU-CSC, then ILU. The, we have ROBO-SGS-4, which is almost as powerful, and then the other ROBO-SGS preconditioner in decreasing order. As expected, Jacobi is the least powerful preconditioner. Regarding efficiency, the most powerful preconditioner is not necessarily the best one. This is discussed in section 7.5.

6.3 Preconditioning in parallel

When doing computations in a parallel hardware environment, a number of things about preconditioners have to be rethought. In particular, GS, SGS and ILU are inherently sequential methods in that some parts of the computations can be carried out only after some other parts. This is not the case for Jacobi and it is also not the case for multigrid methods, provided the smoothers are appropriately chosen. For example, Runge-Kutta smoothers work well in parallel, whereas SGS obviously does not.

However, this behavior can be fixed by using a domain decomposition approach for the preconditioner. This means that the spatial domain is decomposed into several connected subdomains, each of which is assigned to one processor. The preconditioner is then applied locally. When using this approach, care has to be taken since the performance of SGS and ILU in particular depend on the choice of subdomains. The details of how to apply a domain decomposition method will be explained in the next chapter.

6.4 Sequences of linear systems

In the context of unsteady flows, we always have a sequence of linear systems with slowly varying matrices and right hand sides. To obtain an efficient scheme, it is mandatory to exploit this structure. One example for this was the idea of the simplified Newton method, where the Jacobian is not recomputed in every time step. Since setup costs of preconditioners are often high, it is necessary to abstain from computing the preconditioner for every matrix in the sequence. Instead, one of the following strategies should be used.

6.4.1 Freezing and Recomputing

The most common strategy is to compute the preconditioner for the first system and then use it for a number of the following system (freezing), for example by defining a recomputing period l, in that after l time steps, the preconditioner is constructed again. Often, the preconditioner is frozen completely either from the beginning or after a number of time steps [143]. For ILU and Jacobi preconditioning, this is a natural thing, but for SGS, we have another choice. There, it is possible to store only the diagonal and compute the off diagonal blocks on the fly, implying that they have to be computed anew for every system. Alternatively, if storage is not a problem, the off diagonal blocks can be stored, allowing to use freezing and recomputing as a strategy.

6.4.2 Triangular Preconditioner Updates

An idea that aims at reusing information from previous linear systems are preconditioner updates. The technique we base our updates on was suggested originally by Duintjer

Tebbens and Tuma in [52]. It was then reformulated and refined for block matrices [22, 21] and later put into the JFNK context [53]. This strategy has the advantage that it works for any matrix, is easy to implement, parameter-free and with only a small overhead. However, it requires storing not only the matrix and an ILU preconditioner, but also a reference matrix with a reference ILU preconditioner, which means that the storage requirements are high. Nevertheless, it is a powerful strategy if storage is not an issue and has been applied also outside of CFD with good results [218]. This section follows [22].

In addition to a system $\mathbf{Ax} = \mathbf{b}$ with a block ILU(0) (BILU(0)) preconditioner $\mathbf{P} = \mathbf{LDU} = \mathbf{LU_D}$, let $\mathbf{A}^+\mathbf{x}^+ = \mathbf{b}^+$ be a system of the same dimension with the same sparsity pattern arising later in the sequence and denote the difference matrix $\mathbf{A} - \mathbf{A}^+$ by $\mathbf{B}$. We search for an updated block ILU(0) preconditioner $\mathbf{P}^+$ for $\mathbf{A}^+\mathbf{x}^+ = \mathbf{b}^+$. Note that this implies that all matrices have the same sparsity pattern and thus, after a spatial adaptation, the scheme has to be started anew.

We have

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

hence the level of accuracy of $\mathbf{P}^+ \equiv \mathbf{P} - \mathbf{B}$ for $\mathbf{A}^+$ is the same, in the chosen norm, as that of $\mathbf{P}$ for $\mathbf{A}$. The updating techniques from [52] are based on the *ideal* updated preconditioner $\mathbf{P}^+ = \mathbf{P} - \mathbf{B}$. If we would use it as a preconditioner, we would need to solve systems with $\mathbf{P} - \mathbf{B}$ as system matrix in every iteration of the linear solver. For general difference matrices $\mathbf{B}$, these systems would be too hard to solve. Therefore, we will consider cheap approximations of $\mathbf{P} - \mathbf{B}$ instead.

Under the assumption that $\mathbf{P} - \mathbf{B}$ is nonsingular, we approximate its inverse by a product of triangular factors which are easier to invert. In particular, we will end up with using either only the lower triangular or the upper triangular part of $\mathbf{B}$. First, we approximate $\mathbf{P} - \mathbf{B}$ as

$$\mathbf{P} - \mathbf{B} = \mathbf{L}(\mathbf{U_D} - \mathbf{L^{-1}B}) \approx \mathbf{L}(\mathbf{U_D} - \mathbf{B}), \tag{6.10}$$

or by

$$\mathbf{P} - \mathbf{B} = (\mathbf{LD} - \mathbf{BU^{-1}})\mathbf{U} \approx (\mathbf{LD} - \mathbf{B})\mathbf{U}. \tag{6.11}$$

Next we replace $\mathbf{U_D} - \mathbf{B}$ or $\mathbf{LD} - \mathbf{B}$ by a nonsingular and easily invertible approximation. Following [22], we use

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

or

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

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

$$\mathbf{P}^+ = \mathbf{L}(\mathbf{U_D} - btriu(\mathbf{B})) \tag{6.12}$$

and

$$\mathbf{P}^+ = (\mathbf{LD} - btril(\mathbf{B}))\mathbf{U}. \tag{6.13}$$

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

Regarding the distance of the updated preconditioners (6.12) and (6.13) to the ideal preconditioner, we can deduce from the two approximations we make, that it is mainly influenced by the following two properties. The first is closeness of $\mathbf{L}$ or $\mathbf{U}$ to the identity. If matrices have a strong diagonal, the diagonal dominance is in general inherited by the factors $\mathbf{L}$ and $\mathbf{U}$ [15, 14], yielding reasonable approximations of the identity. The second property is a block triangular part containing significantly more relevant information than the other part.

Summarizing, one may expect updates of the form (6.12) or (6.13) to be accurate whenever $btril(\mathbf{B})$ or $btriu(\mathbf{B})$ is a useful approximation of $\mathbf{B}$ and when the corresponding second factor $\mathbf{L}$ or $\mathbf{U}$ is close to the identity matrix. The following lemma from [22] suggests that under the circumstances mentioned, the updates have the potential to be more accurate than the frozen or any other (possibly recomputed) preconditioner for $\mathbf{A}^+$. This lemma is formulated for updates of the form (6.13), where we use the lower part of $\mathbf{B}$; a similar statement can be obtained for updates of the form (6.12).

Lemma 2 *Let* $||\mathbf{A} - \mathbf{LDU}|| = \varepsilon||\mathbf{A}|| < ||\mathbf{B}||$ *for some* $\varepsilon > 0$. *Then the preconditioner from (6.13) satisfies*

$$||\mathbf{A}^+ - \mathbf{P}^+|| \leq \frac{\|\mathbf{U}\| \, \|bstriu(\mathbf{B})\| + ||\mathbf{U} - \mathbf{I}|| \, \|\mathbf{B}\| + \varepsilon||\mathbf{A}||}{||\mathbf{B}|| - \varepsilon||\mathbf{A}||} \cdot ||\mathbf{A}^+ - \mathbf{LDU}||, \quad (6.14)$$

where bstriu denotes the block strict upper triangular part.

This result is a straightforward modification of Lemma 2.1 in [52]. Having a reference preconditioner $\mathbf{LDU}$ which is not too weak we may assume that $\varepsilon||\mathbf{A}||$ is small. Then the multiplication factor before $||\mathbf{A}^+ - \mathbf{LDU}||$ in (6.14) is dominated by the expression $\|\mathbf{U}\| \frac{\|bstriu(\mathbf{B})\|}{\|\mathbf{B}\|} + ||\mathbf{U} - \mathbf{I}||$, which may become smaller than one when $btril(\mathbf{B})$ contains most of $\mathbf{B}$ and when $\mathbf{U}$ is close to the identity matrix. It is possible to show that also the stability of the updates benefits from situations where $btril(\mathbf{B})$ contains most of $\mathbf{B}$ and where $\mathbf{U}$ is close to identity. In our context, the stability is measured by the distance of the preconditioned matrix to the identity. This conforms to the treatment of the stability in [38].

The next result from [22] is more specific to the situation we are interested in here. It presents a sufficient condition for superiority of the update in the case where the frozen preconditioner is a BILU(0) factorization. It is again formulated for the update (6.12), but has, of course, an analogue for (6.13). The matrix $\mathbf{E}$ denotes the error $\mathbf{E} \equiv \mathbf{A} - \mathbf{LDU}$ of the BILU(0) preconditioner and $\| \cdot \|_F$ stays for the Frobenius norm.

Lemma 3 *Let*

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

Then the accuracy $\|\mathbf{A}^+-(\mathbf{LD}-btril(\mathbf{B}))\mathbf{U}\|_F$ *of the updated preconditioner (6.13) is higher than the accuracy of the frozen preconditioner* $\|\mathbf{A}^+-\mathbf{LDU}\|_F^2$ *with*

$$\|\mathbf{A}^+-(\mathbf{LD}-btril(\mathbf{B}))\mathbf{U}\|_F \le \sqrt{\|\mathbf{A}^+-\mathbf{LDU}\|_F^2-(1-\rho)\|btril(\mathbf{B})\|_F^2}. \tag{6.15}$$

Proof: We have, by assumption,

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

Because the sparsity patterns of $\mathbf{B}$ and $\mathbf{E}$ are disjoint,

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

Hence

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

□

With (6.15), the value of ρ may be considered a measure for the superiority of the updated preconditioner over the frozen preconditioner. However, interpretation of the value of ρ is not straightforward. We may write ρ as

$$\rho = \left(\frac{\|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F}{\|btril(\mathbf{B})\|_F}\right)^2 + 2\,\frac{\|\mathbf{E}-bstriu(\mathbf{B})\|_F}{\|btril(\mathbf{B})\|_F^2}, \tag{6.16}$$

where the ratio

$$\frac{\|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F}{\|btril(\mathbf{B})\|_F} \tag{6.17}$$

shows a dependence of ρ on the extent to which $btril(\mathbf{B})$ is reduced after its postmultiplication by $(\mathbf{I}-\mathbf{U})$. This is something slightly different from the dependence of the quality of the update on the closeness of $\mathbf{U}$ to identity. In general, also the second term in (6.16) should be taken into account; only when the lower triangular part clearly dominates and when $\mathbf{LDU}$ is a powerful factorization, one may concentrate on (6.17). Computation of ρ is not feasible in practice because of the expensive product in $\|btril(\mathbf{B})(\mathbf{I}-\mathbf{U})\|_F$ but it offers some insight in what really influences the quality of the update. As the proof of the lemma uses only one inequality, one may expect (6.15) to be a tight bound. We confirm this later by numerical experiments.

We will now describe how to apply updated preconditioners in the solution process. A first issue is the choice between (6.12) and (6.13). We can use some of the previous lemmas to make this choice but we prefer simpler strategies. In [22], three different criterias are suggested.

The first is called the *stable update criterion* and compares the closeness of the factors to identity, keeping the factor that is closest to the identity, which leads to more stable back and forward substitutions. This is done by comparing the norms $\|\mathbf{L} - \mathbf{I}\|$ and $\|\mathbf{U} - \mathbf{I}\|$. If the former norm is smaller, then we update the upper triangular part of the decomposition via (6.12), whereas if, on the contrary, $\mathbf{U}$ is closer to identity in some norm, we update the lower triangular part according to (6.13).

As a second criterion we compare $\|btril(\mathbf{B})\|$ and $\|btriu(\mathbf{B})\|$. We assume the most important information is contained in the dominating block triangular part and therefore we update with (6.12) if $btriu(\mathbf{B})$ dominates $btril(\mathbf{B})$ in an appropriate norm. Otherwise, (6.13) is used. This rule is denoted by *information flow criterion*.

The third criterion is the *unscaled stable update criterion*. It is motivated by considering that using the update (6.12) is straightforward but in order to obtain $\mathbf{U}$ and to apply (6.13) we need to scale $\mathbf{U_D}$ by $\mathbf{D^{-1}}$. Scaling with inverse block diagonal matrices does have, in contrast with inverse diagonal matrices, some influence on overall performance and should be avoided if possible. Note that our stable update criterion compares $\|\mathbf{L}-\mathbf{I}\|$ with $\|\mathbf{U}-\mathbf{I}\|$ where both factors $\mathbf{L}$ and $\mathbf{U}$ have a block diagonal consisting of identity blocks. This means that in order to use the criterion we need to scale $\mathbf{U_D}$, even if the criterion decides for (6.12) and scaling would not have been necessary. This possible inefficiency is circumvented by considering $\mathbf{U_D}$ and $\mathbf{LD}$ instead of $\mathbf{U}$ and $\mathbf{L}$. More precisely, we compare $\|\mathbf{D} - \mathbf{U_D}\|$ with $\|\mathbf{LD} - \mathbf{D}\|$.

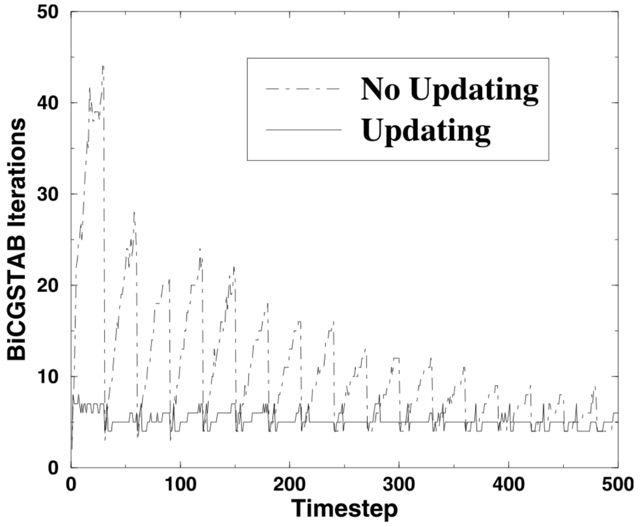
A related issue is the frequency of deciding about the update type based on the chosen criterion. On one hand, there may be differences in the performance of (6.12) and (6.13); on the other hand, switching between the two types implies some additional costs like, for instance, storage of both triangular parts of $\mathbf{B}$. Consequently, the query is used only directly after a recomputation of the BILU(0) decomposition, which takes place periodically. The chosen type of update is then used throughout the whole period. With the information flow criterion we compare $\|btril(\mathbf{B})\|$ with $\|btriu(\mathbf{B})\|$ for the first difference matrix $\mathbf{B}$ generated after recomputation, i.e. just before solving the system following the system for which we used a new BILU(0) decomposition. For the two stable update criteria we may decide immediately which update type should be used as soon as the new BILU(0) decomposition is computed. Note that as soon as the update type is chosen, we need to store only one triangular part of the old reference matrix $\mathbf{A}$ (and two triangular factors of the reference decomposition).

As another tweak, we do not start the updating right away after a recomputation of the frozen preconditioner. This is because in the sequence of linear systems it may happen that several succeeding system matrices are very close and then the frozen preconditioner should be powerful for many subsequent systems. Denote the number of iterations of the linear solver needed to solve the first system of the period by $iter_0$. If for the $(j+1)$st system the corresponding number of iterations $iter_j$ satisfies

$$iter_j > iter_0 + 3, \tag{6.18}$$

where the threshold 3 is chosen heuristically, we start updating.

6.4.3 Numerical results



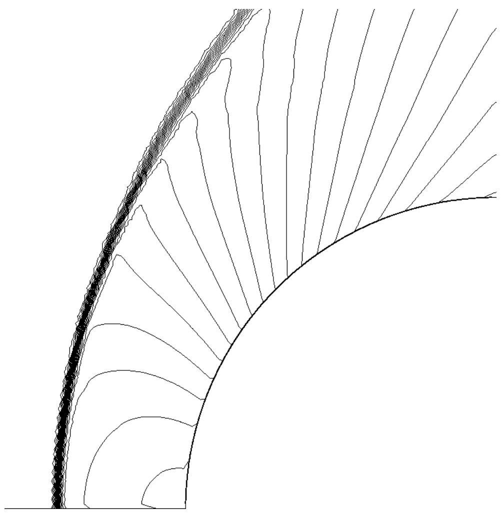


Figure 6.7: BiCGSTAB iterations over timesteps for a cylinder at Mach 10 (left) and pressure isolines (right).

We now compare the triangular updates to freezing and recomputing for a model problem, namely a steady Mach 10 2D Euler flow hitting a cylinder. 3000 steps of the implicit Euler method are performed. In the beginning, a strong shock detaches from the cylinder, which then slowly moves backward through the domain until reaching the steady state position. Therefore, the linear systems are changing only very slowly during the last 2500 time steps and all important changes take place in the initial phase of 500 time steps. The grid consists of 20994 points, whereby only a quarter of the domain is discretized, and system matrices are of dimension 83976. The number of nonzero entries is about $1.33 \cdot 10^6$ for all matrices of the sequence. For the initial data, freestream conditions are used. The initial CFL number is 5, which is increased up to 7 during the iteration. All computations were performed on a Pentium IV with 2.4 GHz. The code used is the in house 2D flow solver TAU_2D. The solution is shown in figure 6.7.

As the flow is supersonic, the characteristics point mostly in one direction and therefore, renumbering leads to a situation where we obtain a matrix with a dominant triangular

i	$\|\mathbf{A}^{(i)} - \mathbf{LDU}\|_F$	$\|\mathbf{A}^{(i)} - \mathbf{P}^{(i)}\|_F$	Bound from (6.15)	ρ from (6.15)
2	37.454	34.277	36.172	0.571
3	37.815	34.475	36.411	0.551
4	42.096	34.959	36.938	0.245
5	50.965	35.517	37.557	0.104
6	55.902	36.118	38.308	0.083

Table 6.1: Accuracy of the preconditioners and theoretical bounds

part. Lemmas 2 and 3 suggest that the updated preconditioner is influenced favorably by this. In figure 6.7, the effect of preconditioner updates on the number of Krylov iterations is compared to a periodic recomputation strategy. As subsequent linear systems change heavily, frozen preconditioners produce rapidly increasing numbers of BiCGSTAB iterations (with decreasing peaks demonstrating the convergence to steady state). Updating, on the other hand, yields a nearly constant number of iterations per time step. The recomputing period here is thirty and the criterion used is the stable update criterion but other periods and criteria give a similar result. With freezing, 5380 BiCGSTAB iterations are performed in this part of the solution process, while the same computation with updating needs only 2611 iterations.

In Table 6.1 we explain the superior performance of the updates with the quantities from Lemma 3 for the very first time steps; they demonstrate the general trend for the whole unsteady phase. Here, $\mathbf{P}^{(i)}$ denotes the update (6.13) for the ith linear system. As the upper bound (6.15) on the accuracy of the updates is very tight, we conclude that in this problem the power of the updates is essentially due to the small values of ρ.

The performance of the updates for the whole sequence is displayed in Table 6.2. To evaluate the results, first note that the reduction of the BiCGSTAB iterations happens primarily in the first 500 time steps. After 500 time steps, freezing is a very efficient strategy and actually gains again on updating. The different updating strategies lead to nearly identical results, whereby the stable update criterion is the best, except for one period. As expected, the update criterions all choose to update the lower triangular part according to (6.13), as the upper triangular part is close to identity due to the numbering of the unknowns and the high Mach number. Therefore, they all obtain the same iteration numbers. Updating is clearly better than freezing if the recomputing period is at least 20. The CPU time is decreased by about 10 % in general; with the recomputing period 50 it reaches up to 20 %. For high recomputing periods, the number of iterations is reduced by even more than 20 %. If the ILU decomposition would have been recomputed in every step, only 11099 BiCGSTAB iterations would be needed, but 28583 seconds of CPU time.

For recomputing periods of 30 or greater, the performance of the updating strategy

	Freezing/Recomp.		Stable update		Unscaled stable update		Information flow	
Period	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s
10	10683	7020	11782	7284	11782	7443	11782	7309
20	12294	6340	12147	6163	12147	6300	12147	6276
30	13787	7119	12503	5886	12503	5894	12503	5991
40	15165	6356	12916	5866	12916	5835	12916	5856
50	16569	6709	13139	5786	13139	5715	13139	5740

Table 6.2: Total iterations and CPU times for supersonic flow example

does not much depend on the period, meaning that the solver becomes more robust in its behavior with respect to the specific choice of the updating period.

When considering different test cases, we see that the triangular updates are less powerful than for this example, if the flows are very steady. Nevertheless, the updating strategy is roughly as efficient as freezing and recomputing, even for test cases where it should perform bad, like steady low Mach number flows.

6.5 Discretization for the preconditioner

The other question, besides which specific preconditioner to use, is, which discretization we base the computation of the blocks on. In the finite volume case, the Jacobian computed is based on the first order discretization in the first place. Now, one advantage of JFNK schemes is that the flux function can be changed easily without a need to reprogram a Jacobian. However, this is no longer the case when an ILU preconditioner is used. Therefore, one idea is to base the computation of the preconditioner on a fixed flux function, thus regaining this advantage of JFNK schemes. In [132], the van Leer flux-vector splitting is used for the preconditioner, as then the blocks can be computed quite efficiently. By contrast, the Jacobian of the AUSMDV or HLLC flux is extremely complicated and expensive to compute.

In figure 6.8, the convergence of ILU(0) preconditioned JFNK-GMRES, where the preconditioner is based on the same flux function as used for the discretization, is compared to the case where a different discretization is used. Furthermore, the results of the scheme with Jacobian are shown. More precise, the in house 2D-FV code TAU_2D is used with the AUSMDV flux and the computation of the Jacobian is based on AUSMDV or the van Leer flux vector splitting. The problem considered is the wind turbine test case A.2 and we show the result of the first time step of an implicit Euler calculation with $CFL = 5$.

As can be seen, the method with mismatched discretizations is only slightly slower than its counterpart. This means that this strategy can be used to simplify the implementation of implicit schemes.

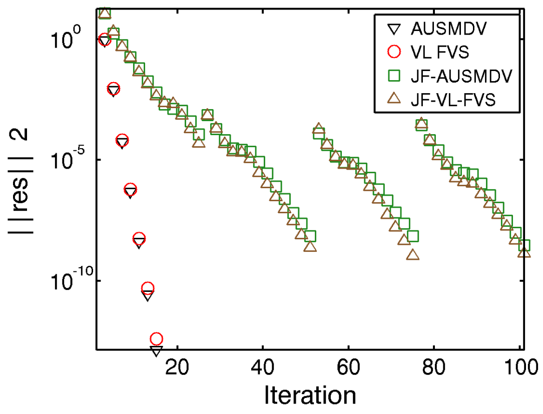


Figure 6.8: Effect of discretization of preconditioner.

Chapter 7

The final schemes

Putting the methodology from the last chapters together, we obtain two prototype schemes, a Rosenbrock type scheme and a DIRK type scheme. In the Rosenbrock type schemes, at each stage of the time integration method, a linear system has to be solved, where the matrix is frozen during a time step, whereas the right hand sides are changing. On the other hand, for the SDIRK type scheme, at each stage of the time integration method, a nonlinear system has to be solved, meaning that we obtain a sequence of linear systems with slowly varying right hand sides and matrices.

All of these schemes need tolerances, termination criterias and parameters. It is imperative to choose these such that accurate schemes are obtained, but it is also important to avoid oversolving and thus obtain efficient schemes. This is rarely discussed in the CFD context, see for example [109] for FV schemes and [104] for DG schemes. We will now collect the results on this documented so far in this book and give reasonable choices that lead to efficient schemes. The core problem here is to find good values for parameters where the existing mathematical results are not guidance enough. In particular, good parameters are problem dependent and different parameters are connected via feedback loops. As an additional goal, there should be as few user defined parameters as possible. At best, we would end up with just one parameter, namely the tolerance for the time integration error error, from which everything else would be determined.

As a further problem, it is difficult to make fair efficiency comparisons of different schemes, because a lot depends on the actual implementation, the test case chosen and also the tolerance we are interested in. It is clear that it is not sufficient to look at the number of Newton iterations needed or the number of time steps chosen. In particular, time step rejections due to a failure in inner convergence change the picture. Also, the impact of a preconditioner cannot be measured in how much it reduces the number of linear iterations, since we have to consider the setup and application cost as well.

For these reasons, we will consider two measures of efficiency, namely the total CPU time and the total number of matrix vector multiplications, which would be equal to the total number of GMRES iterations. Here, total means that we measure this for a complete time integration process from t_0 till t_{end}. The latter indicator is independent of the implementation and of the machine, but does not take into account all the overhead from the Newton iteration, setting up the preconditioner, etc. Furthermore, since GMRES iterations become more costly during one GMRES run, the same number of GMRES iterations for two runs does not mean the same when in a different context of Newton iterations. On the other hand, the CPU time takes these overheads into account, but depends on the machine, the implementation and what else is going on on the machine at a certain point. Nevertheless, the two together give a good assessment of efficiency.

7.1 DIRK scheme

For a DIRK scheme that uses a Newton-Krylov method for the solution of the appearing nonlinear systems, there are three loops where termination criterions and controls need to be defined: The time integration scheme, the nonlinear solver and the linear solver. In the time integration scheme with error tolerance τ, we have the embedded error estimation based on (4.32) and (4.37). The error estimator might lead to a rejection of time steps, but as discussed, this is not necessary. For the method to solve the nonlinear system, termination is tested using either (5.8) or (5.9) with a tolerance, chosen to be $\tau/5$.

As shown in section 5.11, if the time step is chosen too large, Newton's method diverges and the same is true of other methods. Therefore, it is imperative to reject a time step if a tolerance test was failed or if a maximal number of iterations has been reached. In the code SUNDIALS [92], the time step is repeated with $\Delta t_n/4$. Of course, the choice of division by four is somewhat arbitrary, but since the original time step was chosen by the time error estimator, a possible decrease by two will lead to an increase by the factor f_{max} in the next step, which is typically chosen to be two. Therefore, it often happens that in this way, the next time step is rejected again due to the same problems in the nonlinear solver. A division by four leads to less overall rejections and thus to a more efficient scheme.

As an additional tweak, we terminate the computation if the time step size is below a certain threshhold to avoid a computation stalling with ever tinier time steps.

The linear systems are then solved using a preconditioned Jacobian-free GMRES scheme, where the tolerance is chosen by the Eisenstat-Walker strategy, resulting in a second order convergent inexact Newton scheme. This was shown to be the fastest option in section 5.11. If the approximate solution of the linear system still fails the tolerance test after a maximal number of iterations, nothing is done except performing another Newton step.

Another important question is the maximal number of Newton steps allowed before a time step is rejected. To get an idea of a reasonable choice, we use TAU_2D with ESDIRK4, ILU preconditioning and the wind turbine problem A.2 with 10 s of simulation time for different tolerances. As initial time step size, $\Delta t = 7.23349 \cdot 10e{-}4$ is chosen, corresponding to a CFL number of ten. The results are shown in table 7.1.

First of all we can see that there is a large variance in performance with regards to

# Newton Steps	10^{-1}		10^{-2}		10^{-3}	
	Iter.	CPU in s	Iter.	CPU in s	Iter.	CPU in s
15	62,784	24,819	134,365	50,891	139,568	51,139
20	113,782	37,662	97,964	32,675	140,542	45,903
25	114,526	37,373	116,694	36,887	158,018	49,133
30	96,654	28,907	124,390	38,064	157,818	48,449
35	122,387	35,905	177,412	53,714	270,918	80,036
40	139,973	40,201	173,602	49,655	205,378	61,044
45	204,970	57,619	211,810	59,471	206,275	61,197
50	219,252	62,589	279,497	78,129	272,123	79,489

Table 7.1: Effect of different maximal number of Newton iterations on time adaptive ESDIRK4 calculation.

this parameter and that the feedback loop of rejection a time step when the Newton loop does not terminate is highly nonlinear. This is because choosing a large number leads to very large time steps with nonlinear systems that take overly long to solve, whereas a small number in a way leads to a bound on the maximal possible time step. Nevertheless, there are two trends visible. First, allowing a too large maximal number of Newton iterations decreases performance. In addition to the reason just mentioned, this is because at some point, more Newton steps only increase the amount of time spent in diverging Newton iterations. Second, setting this too low is also not a good idea, because time step rejections are very inefficient. We need rejections in case of a diverging Newton methods, but if the maximal number of iterations is too low, this implies that convergent Newton iterations will be cut off causing smaller time steps than necessary. Thus, a number of roughly 30 maximal Newton iterations seems to be reasonable.

All in all, we obtain the following algorithm:

- Given error tolerance τ, initial time t_0 and time step size Δt_0, the number of stages s
- For $n = 0, 1, ...$ do
  - For $i = 1, ..., s$
    - Compute new right hand side
    - Compute initial guess for Newton scheme
    - For $k = 0, 1, ...$ until either (5.8) or (5.9) with tolerance $\tau/5$ is satisfied or k =MAX_NEWTON_ITER

    · Determine relative tolerance for linear system solve via (5.44)
    · Solve linear system using preconditioned GMRES

- If MAX_NEWTON_ITER has been reached, but the tolerance test has not been passed, repeat time step with $\Delta t_n = \Delta t_n/4$
- Estimate local error and compute new time step size Δt_{n+1} using (4.32) and (4.37)
- If $\Delta t_{n+1} < 10^{-20}$, ABORT computation
- $t_{n+1} = t_n + \Delta t_n$

7.2 Rosenbrock scheme

For Rosenbrock schemes, the algorithm becomes easier because there is no Newton scheme and thus one loop less. Nevertheless, we have to define tolerances for the linear systems. Furthermore, similar to the control when the nonlinear solver does not terminate in the SDIRK case, we need to handle the situation when the linear solver does not terminate. Another problem is that with the Rosenbrock methods considered here, it may happen that a result obtained in between or after the time step is unphysical. If this happens, meaning that the norm of the new function evaluation is NaN, we repeat the time step with $\Delta t_n/4$.

Regarding the choice of solver, we have to keep in mind that there is no Newton scheme involved and therefore, the choice of scheme has to be reassessed. To this end, we use the wind turbine test case A.2 for ten seconds of simulation time and ROS34PW2, as well as RODASP with an initial time step of $\Delta t = 3.61674 \cdot 10e - 4$. The linear systems are solved using ILU preconditioned GMRES up to a tolerance of TOL/100. The total number of GMRES iterations and the CPU times can be seen in table 7.2.

First of all, the schemes with a tolerance of 10^{-1} do not converge in the sense that at some time t, the time steps start to get rejected due to NaN appearing and the time integration does not progress anymore. To check if this is due to the linear systems not being solved accurately enough, we repeat the runs and solve the linear systems up to machine accuracy. For ROS34PW2 and the scheme with a first order Jacobian, this leads to an even earlier stall at $t = 200.311s$, whereas the scheme finishes when using Jacobian-free GMRES after 19,266 s of CPU time needing 73,887 GMRES iterations. In the case of RODASP as time integration scheme, solving the linear systems more accurately does not help.

Apparently we are facing a stability problem, since the main difference resulting from the smaller tolerances are the smaller time steps. This could be a loss of A-stability, which occurs in a ROW-method if the approximation of the Jacobian used is too far away from the Jacobian or some additional nonlinear stability problem. Note that in the study on incompressible flows by John and Rang [102], a Jacobian is computed and kept fix during a time step.

		ROS34PW2		RODASP	
Tolerance		1st order Jac.	JF-GMRES	1st order Jac.	JF-GMRES
10^{-1}	Time of stall	200.157	202.157	200.783	205.884
10^{-2}	Iter.	353,421	73,750	368,400	178,361
	CPU in s	81,159	21,467	95,552	52,503
10^{-3}	Iter.	427,484	105,647	Div.	240,744
	CPU in s	105,707	32,923		78,162

Table 7.2: Efficiency of ROS34PW2 and RODASP if the linear systems are solved based on a first order Jacobian or using the Jacobian-free GMRES; linear systems are solved up to TOL/100. For the coarsest tolerance, the time of stall of the time integration scheme is mentioned instead.

Table 7.2 furthermore shows that the Jacobian-free scheme is significantly faster than the scheme with 1st order Jacobian. This deserves further comment. The main reason the JFNK method was faster than the scheme with first order Jacobian in the DIRK context was that the first one acchieves second order convergence with a large radius of convergence, whereas the latter one is only first order convergent with a small radius of convergence. Both of these points are irrelevant for the Rosenbrock scheme. Therefore, two different aspects should come to the fore: The finite difference approximation (5.47) is more expensive than a matrix vector product and GMRES with matrix has a faster convergence behavior. Indeed, after about ten time steps of ROS34PW2 with $TOL = 10^{-2}$, the iteration count of the scheme with first order Jacobian is 326, whereas it is 1643 for the Jacobian-free scheme.

However, since the scheme with first order Jacobian solves the wrong linear system at each stage, it incurs larger time integration errors, leading to an average time step of about 10^{-3} and needing 7,847 time steps in total to reach the final time. Conversely, the scheme with the Jacobian-free matrix-vector multiplication allows an average time step of about 10^{-1} and needs only 124 time steps to reach the final time. This implies that the first order Jacobian is not a good approximation of the second order Jacobian.

Finally, we obtain the following flow diagram for the Rosenbrock type schemes:

- Given error tolerance τ, initial time t_0 and time step size Δt_0, the number of stages s
- For $n = 0, 1, ...$ do
  - For $i = 1, ..., s$
    - Compute new right hand side

    * Estimate initial guess
    * Solve linear system using a preconditioned Krylov subspace method with relative tolerance $\tau/5$.
- If $\|\mathbf{f}(\mathbf{u}_{n+1})\|$ =NaN, repeat time step with $\Delta t_n = \Delta t_n/4$.
- Estimate local error and compute new time step size Δt_{n+1} using (4.33) and (4.37)
- If $\Delta t_{n+1} < 10^{-20}$, ABORT computation
- $t_{n+1} = t_n + \Delta t_n$

7.3 Parallelization

The obvious way of performing parallel computations in the context of the method of lines is to divide the spatial domain into several parts and to assign the computations involving variables from each part to one core. This should be done in a way that minimizes necessary computations betweens domains. Several ways of obtaining such partitions exist and software that does so can be obtained freely, for example the parMETIS-package [107] which is used here.

Thus we use shared memory parallelization and this is done using the MPI standard, with either the MPICH [1] or the openMPI [194] implementation. The parallel computations considered here are not on massively parallel cluster or on GPUs but on standard multicore architectures.

7.4 Efficiency of Finite Volume schemes

To test finite volume schemes, we use the TAU_2D flow solver with the AUSMDV flux and linear reconstruction using the Barth limiter.

In section 5.11, we compared different variants of Newton's method and saw that the JFNK method with the Eisenstat-Walker strategy to determine the tolerances in the GMRES method was by far the most efficient. This does not take into account the case of Rosenbrock schemes. We will now look at the different preconditioners available, where the preconditioner is recomputed every 30 time steps. To this end, we use the time adaptive ESDIRK4 scheme with 30 maximal Newton iterations and a starting time step of $\Delta t = 7.23349 \cdot 10e - 4$, which corresponds to CFL=10. As a test case, we consider again the wind turbine problem A.2 for a time interval of 10 seconds. The results can be seen in table 7.3. As can be seen, ILU reduces the total number of GMRES iterations by a factor of three to six and is two or three times faster than the code without preconditioning. Block Jacobi is only beneficial for larger tolerances and not very powerful, meaning that Jacobi is not a worthwhile preconditioner for finite volume schemes.

Tolerance		None	Jacobi	ILU
10^{-1}	Iter.	319,089	473,692	96,654
	CPU in s	55,580	90,981	28,701
10^{-2}	Iter.	586,013	576,161	124,390
	CPU in s	102,044	111,775	39,056
10^{-3}	Iter.	833,694	642,808	157,818
	CPU in s	142,606	124,901	48,419

Table 7.3: CPU times and iteration numbers for the ESDIRK4 scheme for different preconditioners.

Tolerance		SDIRK2	SDIRK3	ESDIRK3	ESDIRK4
10^{-1}	Iter.	72,707	57,703	68,598	96,654
	CPU in s	21,593	19,041	20,736	28,576
10^{-2}	Iter.	54,622	61,042	89,983	124,390
	CPU in s	17,367	21,699	28,377	38,064
		ROS34PW2	RODASP		
10^{-2}	Iter.	73,750	178,361		
	CPU in s	21,467	52,503		

Table 7.4: CPU times and iteration numbers for different time integration schemes.

We now consider different time integration schemes. To this end, we consider the wind turbine test case for a time interval of 10 seconds and different tolerances. The DIRK schemes all use Jacobian-free GMRES, the Eisenstat-Walker strategy, a maximal number of 30 Newton steps and ILU preconditioning. Furthermore, we consider RODASP and ROS34PW2 with Jacobian-free GMRES, which were shown not to work for the coarse tolerance previously in this chapter. In table 7.4, we can see the total number of GMRES iterations and the CPU times in seconds for the various schemes. The fastest is SDIRK2, then comes SDIRK3, ROS34PW2, ESDIRK3 and ESDIRK4, the worst scheme is RODASP. On the example of SDIRK2, we can see the phenomenon of computational instability [180], meaning that for adaptive schemes, smaller errors not necessarily lead to larger computational cost. In this particular case, SDIRK2 reduces the time step to 10^{-18} via 28 failures to terminate Newton in a row. The reason ESDIRK4 performs so badly is that the time steps chosen are so huge that the nonlinear systems cannot be solved anymore, leading to frequent rejections. The explicit method RK2 needs 40,221 seconds of CPU time.

7.5 Efficiency of Discontinuous Galerkin schemes

7.5.1 Polymorphic Modal-Nodal DG

We first consider the polymorphic modal-nodal method described in section 3.8.1 with different preconditioners. To this end, we use a fixed time integration scheme, namely ESDIRK4, which has proven to be a very good DIRK method. After that, we compare different time integration schemes for the preconditioner found best by the first tests.

As a test case, we use vortex shedding behind a 3D sphere A.3, the code is HALO3D, developed at the University of Stuttgart. The relative and absolute error tolerance in the time integration scheme is 10^{-3} and the nonlinear systems are solved using the Jacobian-Free inexact Newton scheme with the Eisenstat-Walker strategy and GMRES. Since storage is a problem, we allow a maximal Krylov subspace dimension of 20.

For the first time step, we chose $\Delta t = 0.0065$, from then on the steps are determined by the time adaptive algorithm. This time step size was chosen such that the resulting embedded error estimate is between 0.9 and 1, leading to an accepted time step. Then, we perform the computations on a time interval of 30 seconds. The initial solution and the result at the end when using ESDIRK4 time integration with a tolerance of 10^{-3} are shown in figure A.7, where we see isosurfaces of lambda2$=-10^{-4}$, a common vortex identifier [101]. Note that there is no optical difference between the results for ESDIRK4 and the LTSRKCK procedure described in section 4.9.1.

Preconditioner	Iter.	CPU in s	Memory in GB
None	218,754	590,968	16.8
Jacobi	23,369	90,004	17.4
ROBO-SGS-0	18,170	77,316	17.4
ROBO-SGS-1	14,639	66,051	19.0
ROBO-SGS-2	13,446	76,450	24.2
ROBO-SGS-3	13,077	87,115	32.1
ROBO-SGS-4	13,061	100,112	43.8
ILU	12,767	108,031	43.8
ILU-CSC	11,609	127,529	43.8

Table 7.5: Efficiency of preconditioners for DG.

We now compare the use of different preconditioners in table 7.5. The computations were performed in parallel using 8 CPUs. There, the preconditioners are ordered by how powerful we expect them to be, since Jacobi can be seen as the most extreme variant of ROBO-SGS where no information from neighboring cells is taken into account. Thus, as expected from figure 6.6, the total number of GMRES iterations decreases from no preconditioning to Jacobi to ROBO-SGS-0 up to ROBO-SGS-4, then ILU and then ILU-CSC. The first thing that can be see is that by contrast to the finite volume case, Jacobi is a powerful preconditioner. Regarding ROBO-SGS, the decrease is strongest for the first two orders, namely when the constant and the linear part of the solution are taken into account. Consequently, the method of this class with the minimal CPU time is ROBO-SGS-1. Regarding storage, the increase in storage is nonlinear with increasing orders, where ROBO-SGS-4 takes into account the full off diagonal block and thus needs the same amount of storage as ILU, which is more than double as much as ROBO-SGS-1. This demonstrates again the huge size of the blocks of the Jacobian in a DG scheme and why it is so important to find a way of treating these in an efficient way. Furthermore, it explains why Jacobi is actually a good preconditioner.

When comparing ROBO-SGS to ILU, we see that both ILU and ILU with the coarse scale correction are more powerful preconditioners with ILU-CSC being the overall best preconditioner. However, this does not pay in terms of CPU time and thus the overall best preconditioner is ROBO-SGS-1.

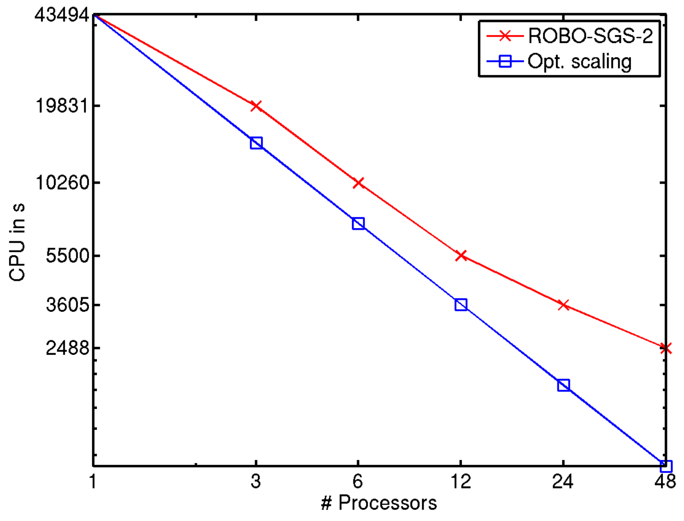


Figure 7.1: Parallel efficiency of ROBO-SGS-1.

Now, we demonstrate the strong parallel scaling of ROBO-SGS-1, meaning the scaling of solution time with the number of processors for a fixed problem size. To this end, we use the initial conditions from above, but compute only until $t_{end} = 133$ and use a fourth order method. This saves storage and allows to use the Twelve-Quad-Core machine from the finite volume part. As can be seen in figure 7.1, the scaling trails off for more than 12 processors. This is roughly the number of processors where the number of unknowns per core drops below 50,000, since we have 740,000 unknowns for this problem. Thus, this behavior is not unexpected.

Scheme	Iter.	CPU in s	Memory in GB
LSERK4	-	346,745	16.3
RKCK	-	80,889	22.9
SDIRK3	22,223	110,844	19.0
ESDIRK3	14,724	73,798	19.0
ESDIRK4	14,639	66,051	19.0
ROS34PW2	60,449	239,869	19.0

Table 7.6: Efficiency of time integration schemes for modal-nodal DG.

Finally, we compare different implicit time integration schemes when using ROBO-SGS-1 as a preconditioner with two explicit methods. The results are shown in table 7.6. As can be seen, the explicit scheme LSERK4 is by far the slowest with about a factor of five between ESDIRK4 and LSERK4. Thus, even when considering that a fourth order explicit scheme is probably overkill for this problem, the message is that explicit Runge-Kutta methods are significantly slower than implicit methods for this test case. Furthermore, the local-time stepping RKCK scheme is about 20% slower than ESDIRK4. Note that for this test case, the time step of the explicit scheme is constrained by the CFL condition and not the DFL condition. Therefore, it can be expected that for higher Reynolds number and finer discretizations at the boundary, the difference in efficiency is even more significant.

When comparing the different implicit schemes, we can see that SDIRK3 is significantly slower than the ESDIRK methods, wheras ESDIRK3 is competitive with ESDIRK4. The worst scheme is the Rosenbrock method, which chooses time steps that are significantly smaller than those of the DIRK methods, which is in line with the experiences from the finite volume case.

7.5.2 DG-SEM

Finally, we consider the second type of DG schemes discussed in the discretization chapter, namely DG-SEM, see 3.8.2. To this end, we consider vortex shedding behind a cylinder in three dimensions at Reynolds number 500 and solve this using the code Strukti, developed at the University of Stuttgart. The result after 1 second of real time for a sixth order scheme on an $8 \times 8 \times 1$ grid can be seen in figure 7.2.

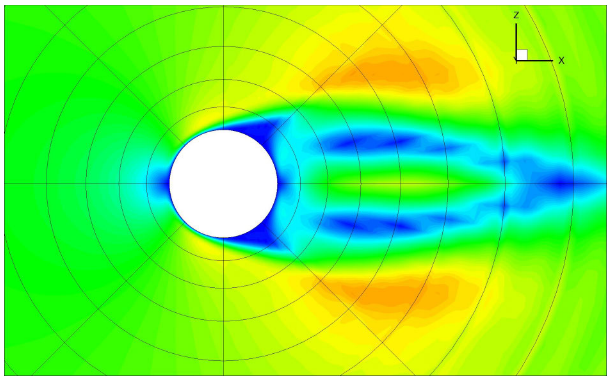


Figure 7.2: $|v|$ around a cylinder at Re 500 after 1 second real time.

To compare efficiency, we use ESDIRK3 with ILU preconditioning and classical Runge-Kutta. The implicit method turns out to be three time slower than the explicit one for a parallel run on 8 processors, even though the time step is 50 times higher. The major difference to the modal-nodal scheme is that due to the nonhierarchic basis, ROBO-SGS cannot be applied. Since the size of the blocks for this scheme is even larger than for the modal-nodal scheme, the class of problems for which implicit schemes are more efficient than explicit ones becomes smaller. Thus, more research needs to be put into finding efficient preconditioners for this type of schemes.

Chapter 8

Thermal Fluid Structure Interaction

One example where the computation of unsteady flows often plays a crucial role is fluid-structure interaction. This can happen in different ways. First of all, aerodynamic forces can lead to structural displacement, leading to different forces and so on. The prime example for this are aeroelastic problems, for example flutter of airfoils or bridges or the performance of wind turbines [59]. Second, heat flux from a fluid can lead to temperature changes in a structure, leading to different heat fluxes and so on.

This is called thermal coupling or thermal fluid structure interaction. Examples for this are cooling of gas-turbine blades, thermal anti-icing systems of airplanes [33] or supersonic reentry of vehicles from space [139, 91]. Another application, which will serve as the main motivating example, is quenching, an industrial heat treatment of metal workpieces. There, the desired material properties are achieved by rapid cooling, which causes solid phase changes, allowing to create graded materials with precisely defined properties.

For the solution of a coupled problem, two general approaches can be distinguished. In a partitioned or staggered approach [59], different codes for the sub-problems are used and the coupling is done by a master program which calls interface functions of the other codes. This allows to use existing software for each sub-problem, in contrast to a monolithic approach, where a new code is tailored for the coupled equations. In the spirit of this book to use flexible components, while still obtaining efficient high order results, we will follow a partitioned approach. We will now explain how the framework of time adaptive higher order implicit methods described so far can be used for efficient solution of thermal coupling problems.

8.1 Gas Quenching

Gas quenching recently received a lot of industrial and scientific interest [171, 89]. In contrast to liquid quenching, this relatively new process has the advantage of minimal environmental impact because of non-toxic quenching media and clean products like air [189]. Furthermore it is possible to control the cooling locally and temporally for best product properties and to minimize distortion by means of adapted jet fields, see [173].

To exploit the multiple advantages of gas quenching the application of computational fluid dynamics has proved essential [6, 189, 126]. Thus, we consider the coupling of the compressible Navier-Stokes equations as a model for air, along a non-moving boundary with the heat equation as a model for the temperature distribution in steel.

8.2 The mathematical model

The basic setting we are in is that on a domain $\Omega_1 \subset \mathbb{R}^d$ the physics is described by a fluid model, whereas on a domain $\Omega_2 \subset \mathbb{R}^d$, a different model describing a structure is used. A model consists of partial differential equations and boundary conditions, as well as possibly additional algebraic equations. The two domains are almost disjoint in that they are connected via an interface. The part of the interface where the fluid and the structure are supposed to interact is called the coupling interface $\Gamma \subset \Omega_1 \cup \Omega_2$, see figure 8.1. Note that Γ might be a true subset of the intersection, because the structure could be insulated. At the coupling interface Γ, coupling conditions are prescribed that model the interaction between fluid and structure. For the thermal coupling problem, these conditions are that temperature and the normal component of the heat flux are continuous across the interface.

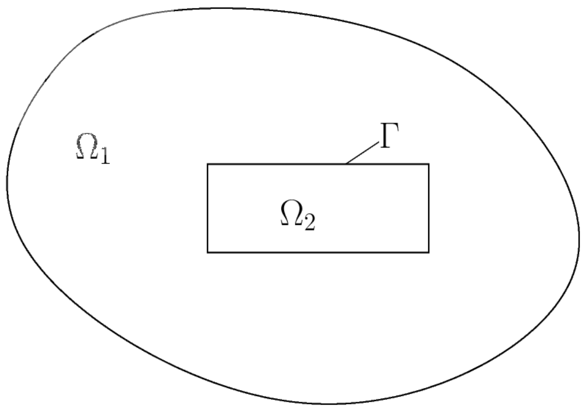


Figure 8.1: Illustration of an FSI problem. The domains Ω_1 and Ω_2 meet at an interface, part of that is the coupling interface Γ.

We model the fluid using the Navier-Stokes equations (2.15), written in compact form as

$$\mathbf{u}_t + \nabla \cdot \mathbf{f}^c(\mathbf{u}) = \nabla \cdot \mathbf{f}^v(\mathbf{u}, \nabla \mathbf{u}). \tag{8.1}$$

Additionally, we prescribe appropriate boundary conditions at all parts of the boundary of Ω_1 except Γ.

Regarding the structure model, heat conduction is derived from Fourier's law, but depending on the application, additional equations for thermomechanical effects could and should be included [88]. Here, we will consider heat conduction only. HIer. For steel, we have temperature-dependent and highly nonlinear specific heat capacity c_D, heat conductivity λ and emissivity. However, we assume that there is no temperature dependence and that heat conductivity is isotropic to obtain the linear heat equation for the structure temperature $\Theta(\mathbf{x}, t)$

$$\rho(\mathbf{x}) c_D \frac{d}{dt} \Theta(\mathbf{x}, t) = -\nabla \cdot \mathbf{q}(\mathbf{x}, t), \tag{8.2}$$

where

$$\mathbf{q}(\mathbf{x}, t) = -\lambda \nabla \Theta(\mathbf{x}, t)$$

denotes the heat flux vector. On the boundary, we have Neumann conditions $\mathbf{q}(\mathbf{x}, t) \cdot \mathbf{n}(\mathbf{x}) = q_b(\mathbf{x}, t)$.

Finally, initial conditions for both domains, $\Theta(\mathbf{x}, t_0) = \Theta_0(\mathbf{x})$, $\mathbf{u}(\mathbf{x}, t_0) = \mathbf{u}_0(\mathbf{x})$ are required.

8.3 Space discretization

Following the partitioned coupling approach, we discretize the two models separately in space. For the fluid, we use a finite volume method, leading to

$$\frac{d}{dt} \mathbf{u}_i + \underline{\mathbf{h}}(\mathbf{u}_i, \mathbf{\Theta}_i) = \mathbf{0}. \tag{8.3}$$

Regarding structural mechanics, the use of finite element methods is ubiquitious. Therefore, we will also follow that approach here and use linear finite elements, leading to the linear equation for all unknowns on Ω_2

$$\mathbf{M} \frac{d}{dt} \mathbf{\Theta}(t) + \mathbf{K} \mathbf{\Theta}(t) - \bar{\mathbf{q}}(\underline{\mathbf{u}}(t)) = \mathbf{0}. \tag{8.4}$$

Here, $\mathbf{M}$ is the heat capacity and $\mathbf{K}$ the heat conductivity matrix. The vector $\mathbf{\Theta}$ consists of all discrete temperature unknowns and $\bar{\mathbf{q}}(\underline{\mathbf{u}}(t))$ is the heat flux vector in the structure.

The coupling conditions have to be enforced implicitely via appropriate boundary conditions on Γ at time t_{n+1}, meaning that we do not know the correct boundary conditions beforehand. Therefore, an iterative Dirichlet-Neumann coupling is employed, as explained in the next section.

8.4 Coupled time integration

A number of articles devoted to the aspect of time integration for fluid-structure interaction problems have been published. In [133], energy conservation for a problem from aeroelasticity is analyzed using the implicit midpoint rule in a monolithic scheme. Already in [13, 43], it is suggested to use an explicit high order Runge-Kutta scheme for both subproblems with data exchange at each stage. Due to the explicit nature, the resulting scheme has severely limited time steps. The order of coupling schemes on moving meshes is analyzed in [77], but only convergence of first order is proved for p-th order schemes. Furthermore, the combination of higher order implicit Runge-Kutta schemes for problems on moving meshes is explored in [209] for the one dimensional case and in the subsequent paper [210] for 3D calculations. There, so called explicit first stage, singly diagonally implicit Runge-Kutta schemes (ESDIRK) are employed and higher order in time is demonstrated by numerical results.

Now, as explained before, if the fluid and the solid solver are able to carry out time steps of implicit Euler type, the master program of the FSI procedure can be extended to SDIRK methods very easily, since the master program just has to call the backward Euler routines with specific time step sizes and starting vectors.

If time adaptivity is added, things become slightly more complicated, because in formula (4.32), all stage derivatives are needed. Therefore, these have to be stored by the master program or the subsolvers. We choose the latter, (4.37) is applied locally and the error estimates are reported back to the master program. Furthermore, if the possibility of rejected time steps is taken into account, $\mathbf{y}_n$ has to be stored as well. Accordingly, the possibilities to achieve higher order accurate results are opened. This approach was first suggested in [28] and the rest of this chapter follows that article.

To comply with the condition that temperature and heat flux are continuous at the interface Γ, a so called Dirichlet-Neumann coupling is used. Namely, the boundary conditions for the two solvers are chosen such that we prescribe Neumann data for one solver and Dirichlet data for the other. Following the analysis of Giles [68], temperature is prescribed for the equation with smaller heat conductivity, here the fluid, and heat flux is given on Γ for the structure. Choosing the conditions the other way around leads to an unstable scheme. Convergence of this approach has been proved for a system of coupled heat equations. However, the case considered here is beyond current convergence theory [160].

In the following it is assumed that the step size Δt_n (or $a_{ii}\Delta t_n$) is prescribed. To show the global procedure at stage i, the starting vector (4.20) of the DIRK-method is decomposed into the fluid and solid variables, $\mathbf{s}_i = \{\mathbf{s}_i^{\mathbf{u}}, \mathbf{S}_i^{\Theta}\}$. According to equation (8.3)-(8.4) the coupled system of equations

$$\mathbf{F}(\mathbf{u}_i, \boldsymbol{\Theta}_i) := \mathbf{u}_i - \mathbf{s}_i^{\mathbf{u}} - \Delta t_n\, a_{ii}\underline{\mathbf{h}}(\mathbf{u}_i, \boldsymbol{\Theta}_i) = \mathbf{0} \tag{8.5}$$

$$\mathbf{T}(\mathbf{u}_i, \boldsymbol{\Theta}_i) := [\mathbf{M} - \Delta t_n\, a_{ii}\mathbf{K}]\boldsymbol{\Theta}_i - \mathbf{M}\mathbf{S}_i^{\Theta} - \Delta t_n\, a_{ii}\bar{\mathbf{q}}(\mathbf{u}_i) = \mathbf{0} \tag{8.6}$$

has to be solved.

The dependence of the fluid equations $\underline{\mathbf{h}}(\mathbf{u}_i, \mathbf{\Theta}_i)$ on the temperature $\mathbf{\Theta}_i$ results from the nodal temperatures of the structure at the interface. This subset is written as $\mathbf{\Theta}_i^\Gamma$. Accordingly, the structure equations depend only on the heat flux of the fluid at the coupling interface.

8.5 Fixed Point iteration

In each stage i the coupled system of nonlinear equations (8.5)-(8.6) has to be solved. Here, a distinction has to be made between loose coupling and strong coupling approaches. In the first approach, only one step of each solver is performed in each time step, while the latter approach adds a convergence criterion and an inner loop. Nonlinear fixed point methods like Block-Gauß-Seidel and Block-Jacobi are typically used here, although also Newton-type methods have been considered [135]. A nonlinear Gauß-Seidel method reads

$$\mathbf{F}(\mathbf{u}_i^{(\nu+1)}, \mathbf{\Theta}_i^{(\nu)}) = \mathbf{0} \quad \rightsquigarrow \quad \mathbf{u}_i^{(\nu+1)} \tag{8.7}$$

$$\mathbf{T}(\mathbf{u}_i^{(\nu+1)}, \mathbf{\Theta}_i^{(\nu+1)}) = \mathbf{0} \quad \rightsquigarrow \quad \mathbf{\Theta}_i^{(\nu+1)} \tag{8.8}$$

within an iteration symbolized by the iteration index (ν). The starting values of the iteration are given by $\mathbf{u}_i^{(0)} = \mathbf{s}_i^{\mathbf{u}}$ and $\mathbf{\Theta}_i^{(0)} = \mathbf{s}_i^{\Theta}$. The termination criterion is formulated by the nodal temperatures at the interface of the solid structure

$$\|\mathbf{\Theta}_i^{\Gamma\,(\nu+1)} - \mathbf{\Theta}_i^{\Gamma\,(\nu)}\| \leq TOL_{FIX}. \tag{8.9}$$

This is interpreted as a fixed point iteration. If the iteration is stopped after on step, e.g. only one computation of (8.7)-(8.8) is carried out, this is also referred to as loose coupling, whereas iterating until the termination criterion is fulfilled corresponds to strong coupling.

Various methods have been proposed to increase the convergence speed of the fixed point iteration by decreasing the interface error between subsequent steps, for example Relaxation [122, 118], Interface-GMRES [144] or ROM-coupling [213]. Relaxation means that after the fixed point iterate is computed, a relaxation step is added:

$$\tilde{\mathbf{\Theta}}_i^{\Gamma\,(\nu+1)} = \omega \mathbf{\Theta}_i^{\Gamma\,(\nu+1)} + (1-\omega)\mathbf{\Theta}_i^{\Gamma\,(\nu)}.$$

Several strategies exist to compute the relaxation parameter ω, in particular Aitken relaxation, which amounts to

$$\omega_{\nu+1} = -\omega_\nu \frac{(\mathbf{r}^{\Gamma\,(\nu)})^T(\mathbf{r}^{\Gamma\,(\nu+1)} - \mathbf{r}^{\Gamma\,(\nu)})}{\|\mathbf{r}^{\Gamma\,(\nu+1)} - \mathbf{r}^{\Gamma\,(\nu)}\|^2}.$$

Here, we use the interface residual

$$\mathbf{r}^{\Gamma\,(\nu)} = \mathbf{\Theta}_i^{\Gamma\,(\nu+1)} - \mathbf{\Theta}_i^{\Gamma\,(\nu)}.$$

8.6 Numerical Results

For the calculations presented here, we will use the DLR TAU-Code, version 2007.1, [66] for the flow and the in-house code TASA-FEM for higher order time integration [86] for the solid. TAU uses a nonlinear multigrid method to solve the appearing nonlinear equation systems, whereas TASA-FEM uses UMFPACK [44] for the appearing linear equation systems. The technical difficulty of different programming languages (FORTRAN for TASA-FEM and C++ for TAU) in the partitioned approach is dealt with a C++-library called Component Template Library (CTL) [134].

8.6.1 Test case

To analyze the properties of the coupling method, we choose a test case that is as simple as possible. The reason is that this comparably simple coupling problem is already beyond the possibilities of current mathematical analysis, although the exact solutions for the uncoupled problems are known. Therefore, to be able to make statements about the error of our scheme, we need to make sure that no additional uncontrollable side effects are present.

Accordingly, the cooling of a flat plate resembling a simple work piece is considered. This example has also been studied by other authors [224] and [95, p. 465] in conjunction with the cooling of structural parts in hypersonic vehicles. There, localized heating was of special interest. In our case the work piece is initially at a much higher temperature than the fluid and then cooled by a constant air stream. The latter is modeled in a first approximation as a laminar flow along the plate, see figure 8.2.

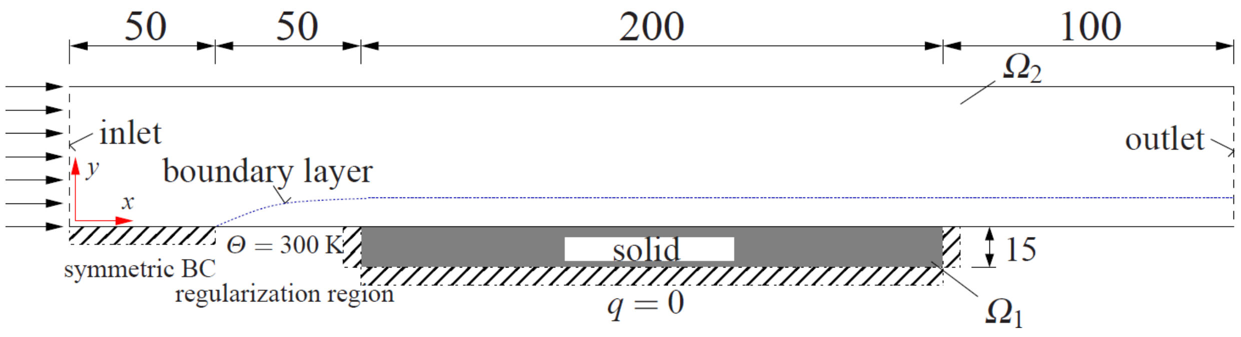


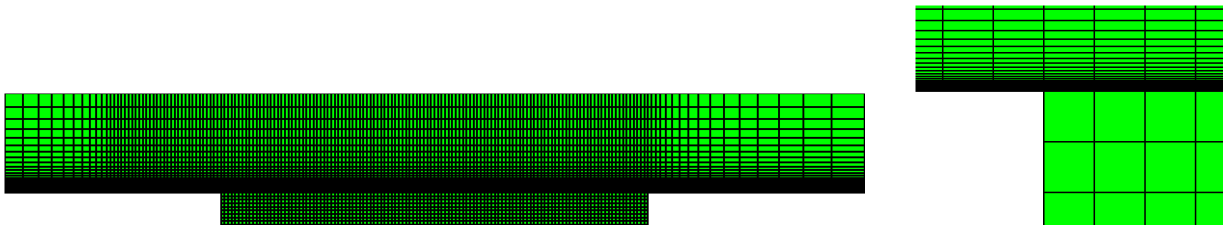
Figure 8.2: Test case for the coupling method

The inlet is given on the left, where air enters the domain with an initial velocity of $\mathrm{Ma}_\infty = 0.8$ in horizontal direction and a temperature of 273 K. Then, there are two succeeding regularization regions of 50 mm to obtain an unperturbed boundary layer. In the first region, $0 \leq x \leq 50$, symmetry boundary conditions, $v_y = 0$, $q = 0$, are applied. In the second region, $50 \leq x \leq 100$, a constant wall temperature of 300 K is specified. Within this region the velocity boundary layer fully develops. The third part is the solid (work piece) of length 200 mm, which exchanges heat with the fluid, but is assumed insulated otherwise, $q = 0$. Therefore, Neumann boundary conditions are applied throughout. Finally, the fluid domain is closed by a second regularization region of 100 mm with symmetry boundary conditions and the outlet.

As material in the structure we use steel, where the following constant material properties are assumed: mass density $\rho = 7836$ kg/m^3, specific heat capacity $c_D = 443$ J/(kgK) and thermal conductivity $\lambda = 48.9$ W/(mK).

Regarding the initial conditions in the structure, a constant temperature of 900 K at $t = 0$ s is chosen throughout. To specify reasonable initial conditions within the fluid, a steady state solution of the fluid with constant wall temperature $\Theta = 900$ K is computed. Thus, we are able to compare the results with the theoretical solution of Blasius for the velocity boundary layer and experimental results of van Driest for the temperature boundary layer [206]. Furthermore, the grid of the fluid is refined manually until a good agreement with these is achieved.

The grid is chosen cartesian and equidistant in the structural part, where in the fluid



(a) Entire mesh (b) Mesh zoom

Figure 8.3: Full grid (left) and zoom into coupling region (right)

region the thinnest cells are on the boundary and then become coarser in y-direction (see figure 8.3). To avoid additional difficulties from interpolation, the points of the primary fluid grid, where the heat flux is located in the fluid solver, and the nodes of the structural grid are chosen to match on the interface Γ.

8.6.2 Order of the method

Figure 8.4 shows the temporal evolution of the temperature at two points of the coupling interface, namely the beginning ($x = 100$) and the end ($x = 300$) of the coupling interface. The abbreviation BE symbolizes the Backward-Euler method, whereas El corresponds to the second order method of Ellsiepen, SDIRK2. As expected, the temperature decreases monotonously with a large gradient in the beginning which then becomes smaller. At $t = 1$ s, the temperature in the end point has dropped to 895 K. Furthermore, the temperature decrease at the beginning of the solid part is much more pronounced than further downstream, where the gas has already been heated. In other words, the heat absorption is strongly inhomogeneous from the left to the right.

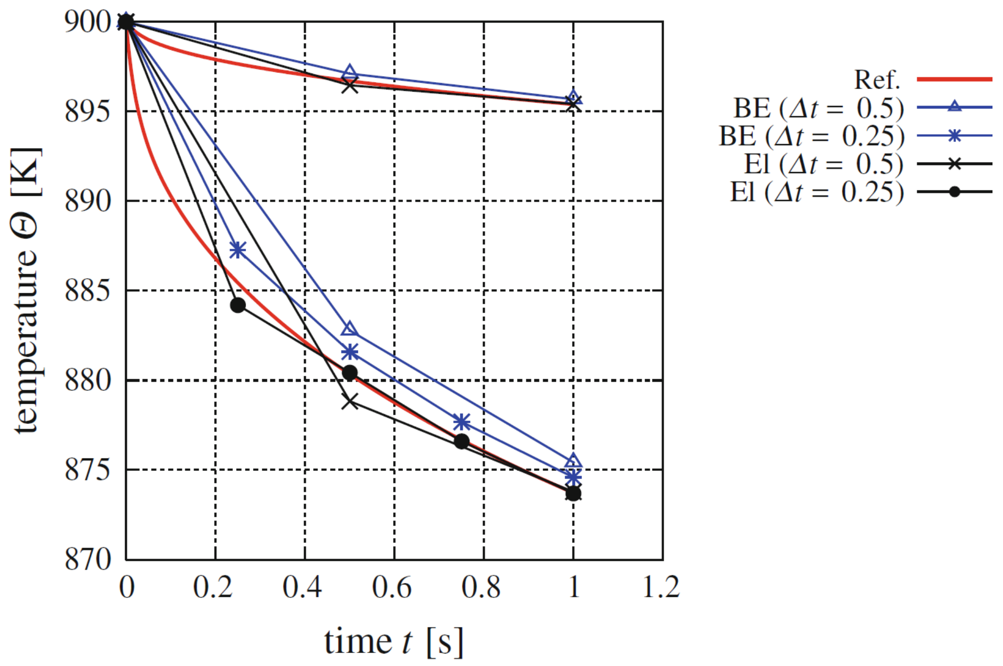


Figure 8.4: Temperature evolution at interface for different methods and time step sizes: BE is backward Euler, El Ellsiepens method and no index the reference solution. The upper curves correspond to the endpoint, the lower curves to the first point of the coupling interface.

A first computation is done on a fine grid with 54600 cells in the fluid region and 6934 in the structure domain, whereas a second computation (coarse grid) is done with 9660 cells in the fluid domain and $n_x \times n_y = 120 \times 9 = 1080$ elements with $121 \times 10 = 1210$ nodes for the structure.

Since no exact solution is available, we compute reference solutions using a time step of $\Delta t_n = 0.001$ s. The fine grid solution is necessary to show that the coarse grid solution is reasonable. Both solutions are obtained using the implicit Euler method using fixed point coupling and a fixed tolerance (see next paragraph) for all involved solvers.

In the following the implicit Euler method is compared to SDIRK2. Figure 8.5 shows the error compared to the reference solution for different time step sizes. All tolerances (within the nonlinear multigrid solver of the fluid and the fixed point iteration of equation (8.9)) are set to $TOL = 10^{-7}$. BE-FPGS and EL-FPGS, respectively, correspond to the implicit Euler and SDIRK2 combined with the fixed point iteration in equation (8.9).

Obviously, the implicit Euler method is of first order, whereas SDIRK2 is of higher order. The second order is clearly seen if strong coupling is used, whereas using loose coupling results in a decrease of the order to one.

Regarding computational effort, it is useful to measure this by the number of fixed

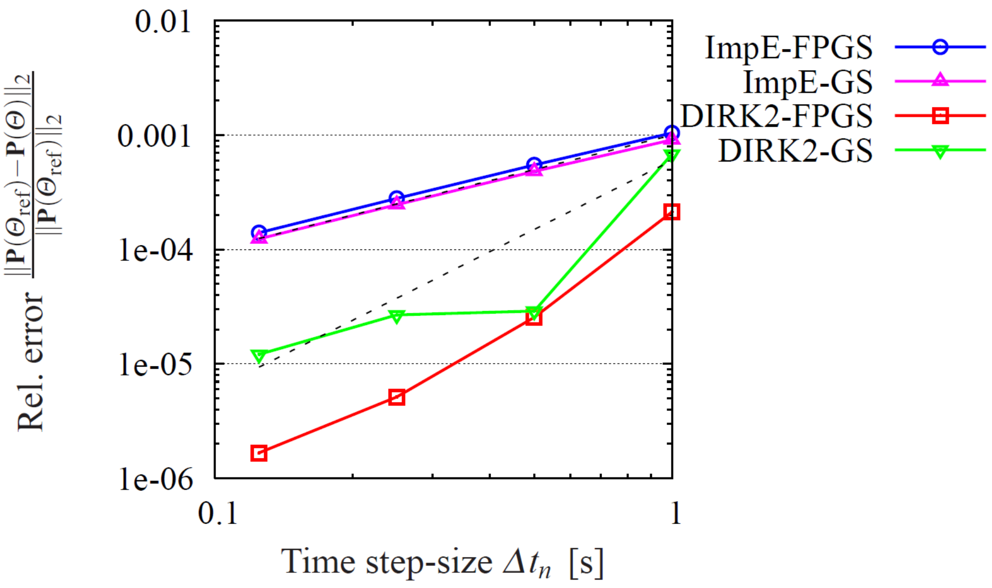


Figure 8.5: Order diagram for coarse and fine grid

point iterations needed. The El-GS method (combination of SDIRK2 and one Gauß-Seidel step) needs exactly twice as much as the BE-GS method (implicit Euler method with one Gauß-Seidel step) since both take by definition only one fixed point iteration per stage.

In view of the fully converged fixed point solvers, SDIRK2 needs slightly less than twice the effort of the implicit Euler solver. This is due to the decreased time-step in the intermediate stages, which accounts for slightly simpler FSI-problems and, consequently, faster convergence of the fixed point algorithms. Note that there is a fair number of methods to accelerate convergence of the fixed point solver, which are not discussed here.

The main computational effort is in the fluid solver, for two reasons. First of all, the number of grid cells is significantly larger than in the solid problem and there are four unknowns per grid cell leading to a much larger system. Furthermore, the strong temperature boundary layer leads to a significant loss of speed of convergence in the fluid solver, which is the bottleneck in the first place.

8.6.3 Time-adaptive computations

To test the time-adaptive method, the test case from above on the coarse grid is used again. For the iterations, namely the fixed point iteration of the coupled system (8.9) and the multigrid method in the fluid, we use a tolerance which is ten times smaller than ϵ of the local error estimation. Here, $\epsilon := ATOL = RTOL$ is chosen. Results for $t_{end} = 1$ s with two local error tolerances $\epsilon_1 = 10^{-3}$ and $\epsilon_2 = 10^{-4}$ are compared with the reference solution from the last section to make sure that the error estimator works properly.

In the first case of $\epsilon_1 = 10^{-3}$ the error estimator says that a time-step of 1 s is sufficient to satisfy the tolerance, which is confirmed by the results from the last section. For the second case, $\epsilon_2 = 10^{-4}$, the error estimator suggests to start with $\Delta t_n = 0.15$, which is below the $\Delta t_n = 0.5$ obtained by the results from the last section and, accordingly, is slightly too conservative.

Finally, a computation until $t_{end} = 30$ s is carried out to compare efficiency of the time-adaptive approach with a fixed time-step computation using $\Delta t_n = 1$ s. The resulting temperature isolines can be seen in figure 8.6, indicating that there is a strongly inhomo-

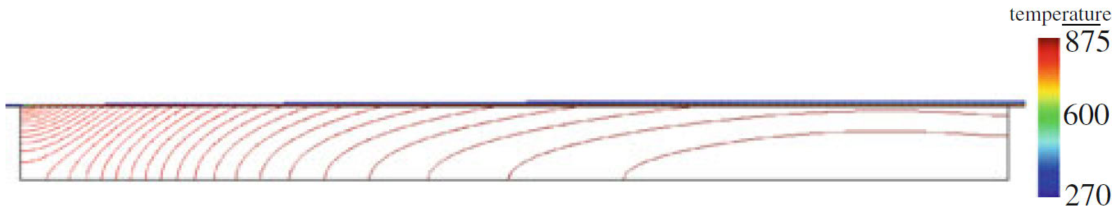


Figure 8.6: Iso-lines of temperature at $t = 30$ s

geneous heat transport to the fluid from the solid in horizontal direction. Additionally, the evolving boundary layer becomes obvious. This is also depicted in greater detail in figure 8.7, where the cross section of the temperature profile at the beginning ($x = 100$) and the end ($x = 200$) of the solid is shown.

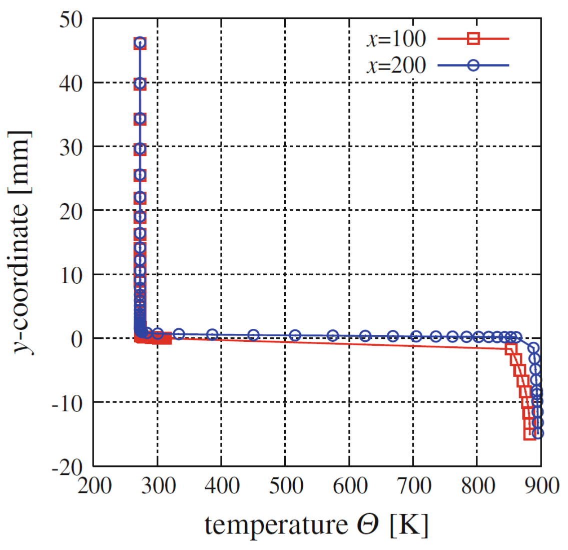


Figure 8.7: Temperature distribution in cross sections at $t = 30$ s

As starting step size the one from the last computations is chosen. The temperature evolution at point $x = 100$ of the interface for the three different computations (fixed step size $\Delta t_n = 1$ s, tolerance $\epsilon_1 = 10^{-3}$, and tolerance $\epsilon_2 = 10^{-4}$) can be seen in figure 8.8. Note that the time step control always increases the step size in the problem under consideration,

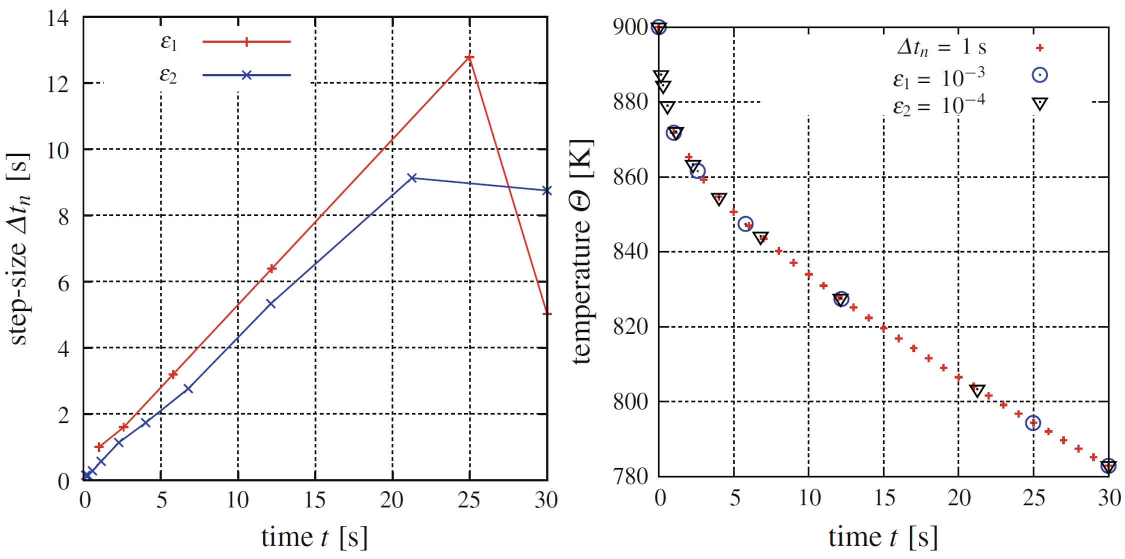


Figure 8.8: Step-size behavior of the entire thermal FSI-problem (left) and temperature evolution at interface (right).

where no external changes influence the evolution of the transient process, see figure 8.8 (left). However, for a low tolerance the step size estimation is much more restrictive.

104 fixed point iterations are needed in total for the computation with ϵ_2 and 63 for the computation with ϵ_1. If the fixed time step size is employed, 243 are needed. Since they have the same starting step size, the low tolerance solution is less accurate than the fixed time-step solution, but the strong tolerance solution is more accurate than the fixed time step solution. However, the fixed time step solution needs more than twice as much fixed point iterations as the strong tolerance solution. Thus, the adaptive method is more than twice as efficient as the fixed time step-size method for this problem.

Appendix A

Test problems

A.1 Shu-Vortex

For the Euler equation, a 2D problem where the exact solution is known is the convection of a 2D vortex in a periodic domain, which is designed to be isentropic [45]. The initial conditions are taken as freestream values $(\rho_\infty, v_{1_\infty}, v_{2_\infty}, T_\infty) = (1, v_{1_\infty}, 0, 1)$, but are perturbed at t_0 by a vortex $(\delta v_1, \delta v_2, \delta T)$ centered at $(\tilde{x}_1, \tilde{x}_2)$, given by the formulas

$$\begin{aligned}
\delta v_1 &= -\frac{\alpha}{2\pi}(x_2 - \tilde{x}_2)e^{\phi(1-r^2)}, \\
\delta v_2 &= \frac{\alpha}{2\pi}(x_1 - \tilde{x}_1)e^{\phi(1-r^2)}, \\
\delta T &= -\frac{\alpha^2(\gamma-1)}{16\phi\gamma\pi^2}(x_1 - \tilde{x}_1)e^{2\phi(1-r^2)},
\end{aligned} \tag{A.1}$$

where ϕ and α are parameters and r is the euclidian distance of a point (x_1, x_2) to the vortex center, which is set at $(\tilde{x}_1, \tilde{x}_2) = (0, 0)$ in the domain $[-7, 7] \times [-3.5, 3.5]$. The parameter α can be used to tweak the speed of the flow in the vortex, whereas ϕ determines the size. The initial solution can be seen in figure A.1.

The functions (A.1) do not provide an exact solution of the Navier-Stokes equations. There, we have additional diffusion, which leads to a decay of the solution in time. To test our methods, we use a structured grid with 23500 cells, which is illustrated in figure A.2.

A.2 Wind Turbine

The second test case is a two dimensional flow around the cross section of the wind turbine DUW-96, designed by the wind energy research initiative DUWIND of the TU Delft. The

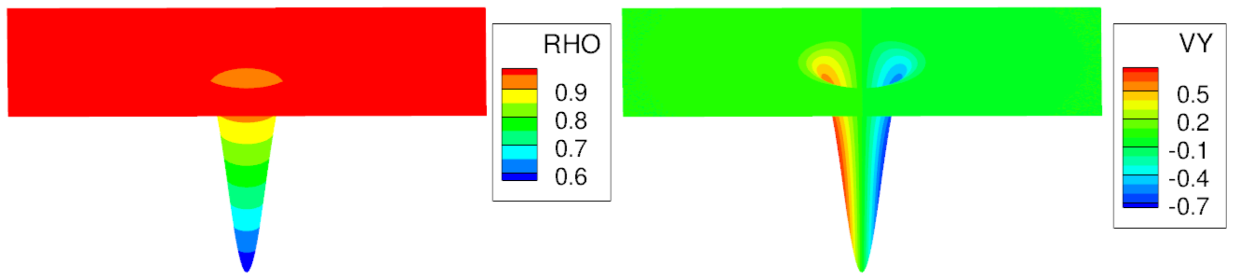


Figure A.1: Initial solution of isentropic vortex solution of the Euler equations. Density (left) and v_y (right).

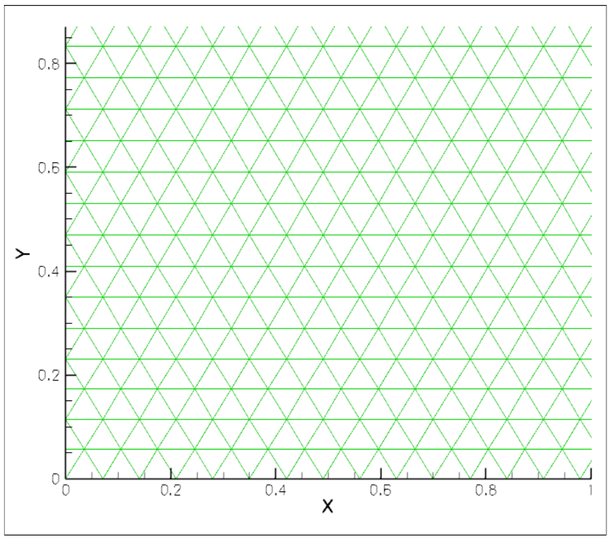


Figure A.2: Excerpt of grid for Shu vortex problem.

Mach number is 0.12 with an angle of attack of 40° and the Reynolds number 1,000. The grid has 24,345 cells and is illustrated in figure A.3.

To obtain an unsteady test case, we first compute the steady state around this for an Euler flow. When starting from this solution, an immediate vortex shedding starts, which is slow due to the low Reynolds number. The steady state solution and the ones after 30, 60 and 90 seconds of simulation time can be seen in figures A.4 and A.5. These were computed using SDIRK2 at a tolerance of 10^{-2}.

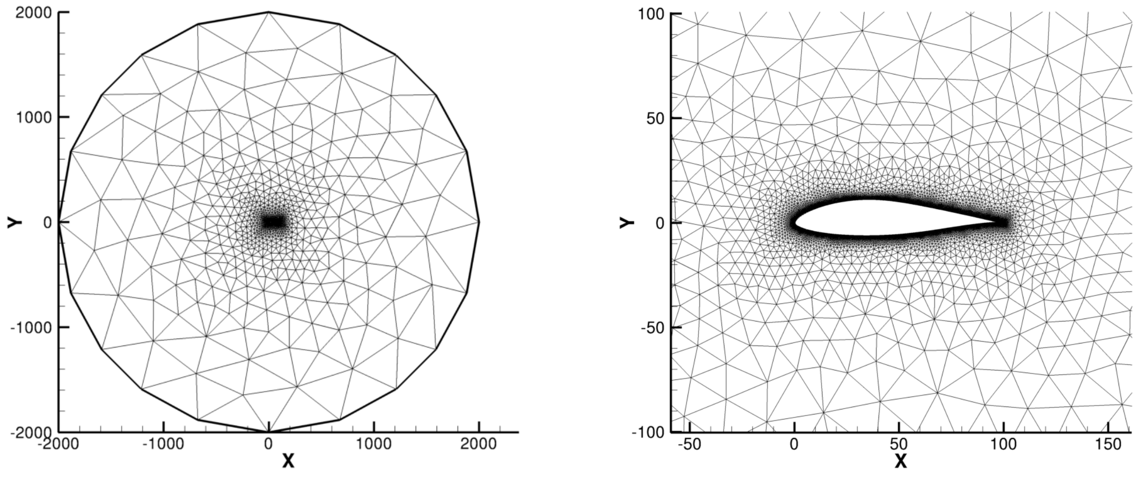


Figure A.3: Grid around wind turbine.

A.3 Vortex shedding behind a sphere

This test case is a three dimensional test case about vortex shedding behind a sphere at Mach 0.3 and a Reynolds number of 1,000. The grid is illustrated in figure A.6. It consists of 21,128 hexahedral cells.

To obtain initial conditions and start the initial vortex shedding, we begin with free stream data and with a second order method. Furthermore, to cause an initial disturbance, the boundary of the sphere is chosen to be noncurved at first. In this way, we compute 120 s of time. Then we switch to fourth order, compute another 10 s of time. From this, we start the actual computations with a 5th order method and a curved representation of the boundary. Thus, we have 739,480 unknowns.

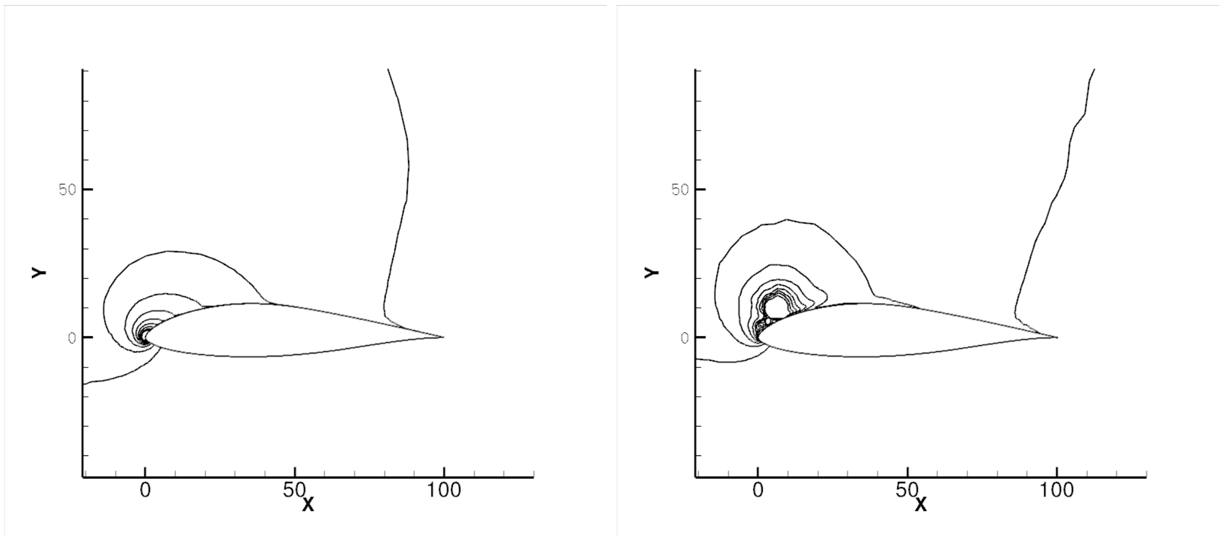


Figure A.4: Density isolines for wind turbine problem at $t = 0$s and $t = 30$s.

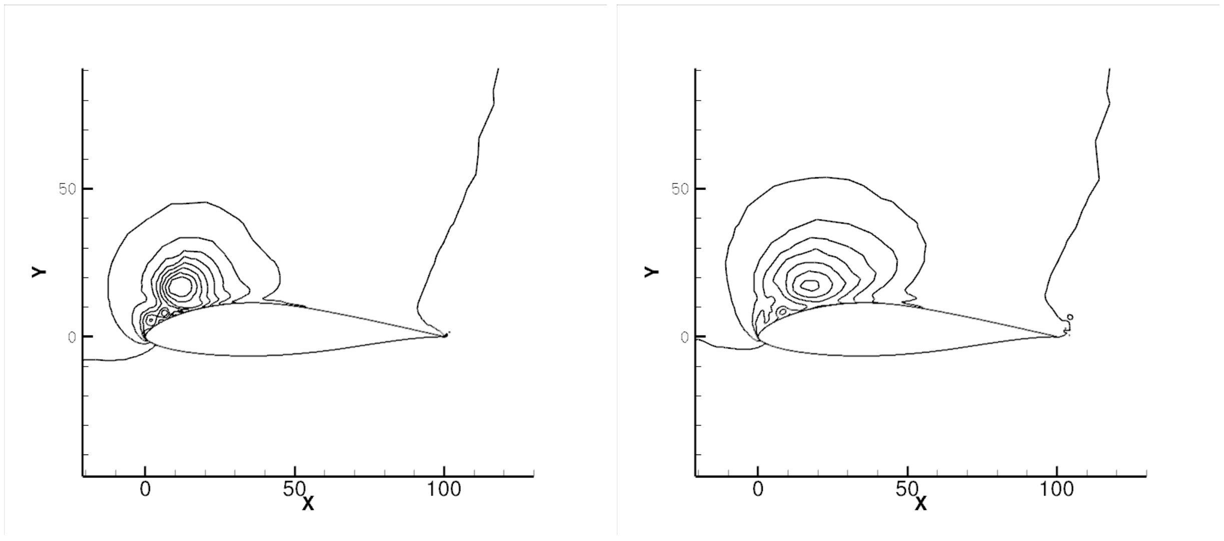


Figure A.5: Density isolines for wind turbine problem at $t = 60$s and $t = 90$s.

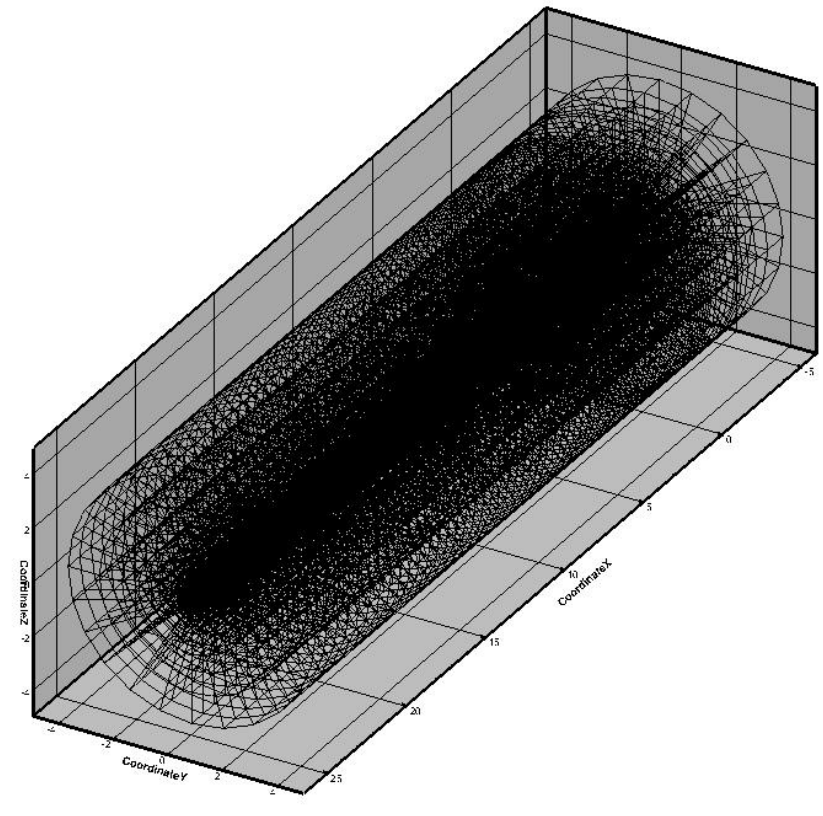


Figure A.6: Grid for sphere test case.

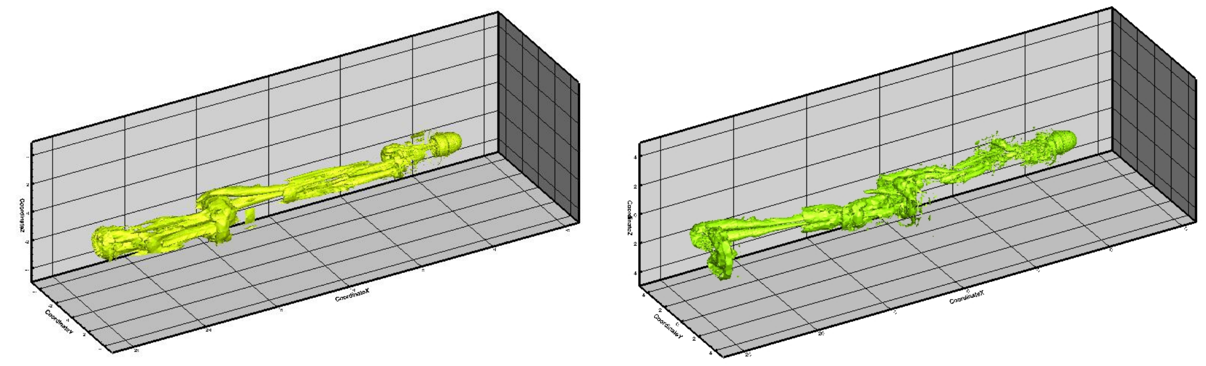


Figure A.7: Isosurfaces of lambda-2=-10^{-4} for initial (left) and final solution of sphere problem (right).

Appendix B

Coefficients of time integration methods

1	1	
	1/2	1/2

1	1		
1/2	1/4	1/4	
	1/6	1/6	2/3

Table B.1: Explicit SSP methods: SSP2 and SSP3

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	1/3	1/3	1/6

Table B.2: Coefficients for RK4

i	a_i	b_i
1	0	$\frac{1432997174477}{9575080441755}$
2	$-\frac{567301805773}{1357537059087}$	$\frac{5161836677717}{13612068292357}$
3	$-\frac{2404267990393}{2016746695238}$	$\frac{1720146321549}{2090206949498}$
4	$-\frac{3550918686646}{2091501179385}$	$\frac{3134564353537}{4481467310338}$
5	$-\frac{1275806237668}{842570457699}$	$\frac{2277821191437}{14882151754819}$

Table B.3: LSERK4 coefficients

α	α	0
1	$1-\alpha$	α
b_i	$1-\alpha$	α
$\hat{b}_i$	$1-\hat{\alpha}$	$\hat{\alpha}$
$b_i - \hat{b}_i$	$\hat{\alpha}-\alpha$	$\alpha-\hat{\alpha}$

$$\alpha = 1 - \sqrt{2}/2$$
$$\hat{\alpha} = 2 - \tfrac{5}{4}\sqrt{2}$$
$$\alpha - \hat{\alpha} = -1 + \tfrac{3}{4}\sqrt{2}$$

Table B.4: Butcher array for the method of Ellsiepen.

γ	γ	0	0
δ	$\delta-\gamma$	γ	0
1	α	β	γ
b_i	α	β	γ
$\hat{b}_i$	$\hat{\alpha}$	$\hat{\beta}$	0
$b_i - \hat{b}_i$			γ

$$\alpha = 1.2084966491760101$$
$$\beta = -0.6443631706844691$$
$$\gamma = 0.4358665215084580$$
$$\delta = 0.7179332607542295$$
$$\delta - \gamma = 0.2820667392457705$$
$$\hat{\alpha} = 0.7726301276675511$$
$$\hat{\beta} = 0.2273698723324489$$

Table B.5: Butcher array for the method of Cash.

0	0	0	0	0
2γ	γ	γ	0	0
3/5	0.257648246066427	-0.093514767574886	γ	0
1	0.187641024346724	-0.595297473576955	0.971789927721772	γ
b_i	0.187641024346724	-0.595297473576955	0.971789927721772	γ
$\hat{b}_i$	0.214740286223389	-0.485162263884939	0.868725002520388	0.401696975141162
$b_i - \hat{b}_i$	-0.027099261876665	-0.110135209692016	0.103064925201385	0.034169546367297

Table B.6: Butcher diagram for ESDIRK3. $\gamma = 0.435866521508459$

0	0	0	0	0	0	0
2γ	γ	γ	0	0	0	0
83/250	0.137776	-0.055776	γ	0	0	0
31/50	0.144636866026982	-0.223931907613345	0.449295041586363	γ	0	0
17/20	0.098258783283565	-0.591544242819670	0.810121053828300	0.283164405707806	γ	0
1	0.157916295161671	0	0.186758940524001	0.680565295309335	-0.275240530995007	γ
b_i	0.157916295161671	0	0.186758940524001	0.680565295309335	-0.275240530995007	γ
$\hat{b}_i$	0.154711800763212	0	0.189205191660680	0.702045371228922	-0.319187399063579	0.273225035410765
$b_i - \hat{b}_i$	0.003204494398459	0	-0.002446251136679	-0.021480075919587	0.043946868068572	-0.023225035410765

Table B.7: Butcher diagram for ESDIRK4. $\gamma = 0.25$

j i	1	2	3	4
1	0.203524508852533	-1.888641733544525	2.596814547851191	0.088302676840802
2	-0.015883484505809	1.29334425996757	-1.625024620129419	0.347563844667657

Table B.8: Coefficients b^*_{ij} of the dense output formulas for ESDIRK3

j i	1	2	3	4	5	6
1	0.924880883935497	0	0.73854539473069	-2.517224551339271	3.5887741459555	-1.734975873282416
2	-0.766964588773826	0	-0.551786454206689	3.197789846648606	-3.864014676950506	1.984975873282416

Table B.9: Coefficients b^*_{ij} of the dense output formulas for ESDIRK4

$\gamma = 0.435866521508459$			
$\tilde{\alpha}_{21} = 0.87173304301691801$	$\gamma_{21} = -0.87173304301691801$	$a_{21} = 2$	$c_{21} = -4.588560720558085$
$\tilde{\alpha}_{31} = 0.84457060015369423$	$\gamma_{31} = -0.90338057013044082$	$a_{31} = 1.419217317455765$	$c_{31} = -4.184760482319161$
$\tilde{\alpha}_{32} = -0.11299064236484185$	$\gamma_{32} = 0.054180672388095326$	$a_{32} = -0.259232211672970$	$c_{32} = 0.285192017355496$
$\tilde{\alpha}_{41} = 0$	$\gamma_{41} = 0.24212380706095346$	$a_{41} = 4.184760482319161$	$c_{41} = -6.36817920012836$
$\tilde{\alpha}_{42} = 0$	$\gamma_{42} = -1.2232505839045147$	$a_{42} = -0.285192017355496$	$c_{42} = -6.795620944466837$
$\tilde{\alpha}_{43} = 1$	$\gamma_{43} = 0.54526025533510214$	$a_{43} = 2.294280360279042$	$c_{43} = 2.870098604331056$
$b_1 = 0.24212380706095346$	$\hat{b}_1 = 0.37810903145819369$	$m_1 = 4.184760482319160$	$\hat{m}_1 = 3.907010534671193$
$b_2 = -1.2232505839045147$	$\hat{b}_2 = -0.096042292212423178$	$m_2 = -0.285192017355497$	$\hat{m}_2 = 1.118047877820503$
$b_3 = 1.545260255335102$	$\hat{b}_3 = 0.5$	$m_3 = 2.294280360279042$	$\hat{m}_3 = 0.521650232611491$
$b_4 = 0.435866521508459$	$\hat{b}_4 = 0.2179332607542295$	$m_4 = 1$	$\hat{m}_4 = 0.5$

Table B.10: Coefficients for ROS34PW2

$\gamma = 0.25$			
$\tilde{\alpha}_{21} = 0.75$	$\gamma_{21} = -0.75$	$a_{21} = 3.0$	$c_{21} = -12.0$
$\tilde{\alpha}_{31} = 0.08612040081415375$	$\gamma_{31} = -0.13551240081415375$	$a_{31} = 1.831036793486770$	$c_{31} = -8.791795173947078$
$\tilde{\alpha}_{32} = 0.12387959918584625$	$\gamma_{32} = -0.13799159918584625$	$a_{32} = 0.495518396743385$	$c_{32} = -2.207865586973539$
$\tilde{\alpha}_{41} = 0.77403453550732462$	$\gamma_{41} = -1.2560840048950797$	$a_{41} = 2.304376582692655$	$c_{41} = 10.817930568571748$
$\tilde{\alpha}_{42} = 0.14926515495086924$	$\gamma_{42} = -0.25014471050642479$	$a_{42} = -0.052492752457437$	$c_{42} = 6.780270611428374$
$\tilde{\alpha}_{43} = -0.29419969045819386$	$\gamma_{43} = 1.2209287154015045$	$a_{43} = -1.176798761832776$	$c_{43} = 19.534859446424075$
$\tilde{\alpha}_{51} = 5.308746682646143$	$\gamma_{51} = -7.0731843314206279$	$a_{51} = -7.170454962423204$	$c_{51} = 34.190950067497354$
$\tilde{\alpha}_{52} = 1.3308921400372693$	$\gamma_{52} = -1.8056486972435724$	$a_{52} = -4.741636671481872$	$c_{52} = 15.496711537259923$
$\tilde{\alpha}_{53} = -5.3741378116555623$	$\gamma_{53} = 7.7438296585713671$	$a_{53} = -16.310026313309713$	$c_{53} = 54.747608759641373$
$\tilde{\alpha}_{54} = -0.26550101102784999$	$\gamma_{54} = 0.88500337009283331$	$a_{54} = -1.062004044111401$	$c_{54} = 14.160053921485337$
$\tilde{\alpha}_{61} = -1.7644376487744849$	$\gamma_{61} = 1.6840692779853733$	$a_{61} = -7.170454962423204$	$c_{61} = 34.626058309305947$
$\tilde{\alpha}_{62} = -0.47475655720630317$	$\gamma_{62} = 0.41826594361385614$	$a_{62} = -4.741636671481872$	$c_{62} = 15.300849761145031$
$\tilde{\alpha}_{63} = 2.3696918469158048$	$\gamma_{63} = -1.881406216873008$	$a_{63} = -16.310026313309713$	$c_{63} = 56.999555786626757$
$\tilde{\alpha}_{64} = 0.61950235906498332$	$\gamma_{64} = -0.11378614758336428$	$a_{64} = -1.062004044111401$	$c_{64} = 18.408070097930938$
$\tilde{\alpha}_{65} = 0.25$	$\gamma_{65} = -0.35714285714285714$	$a_{65} = 1.0$	$c_{65} = -5.714285714285714$
$b_1 = -0.08036837078911165$	$\hat{b}_1 = -1.7644376487744849$	$m_1 = -7.170454962423198$	$\hat{m}_1 = -7.1704549624232$
$b_2 = -0.056490613592447036$	$\hat{b}_2 = -0.47475655720630317$	$m_2 = -4.741636671481874$	$\hat{m}_2 = -4.741636671481875$
$b_3 = 0.48828563004279679$	$\hat{b}_3 = 2.3696918469158048$	$m_3 = -16.310026313309713$	$\hat{m}_3 = -16.310026313309713$
$b_4 = 0.50571621148161904$	$\hat{b}_4 = 0.61950235906498332$	$m_4 = -1.062004044111401$	$\hat{m}_4 = -1.062004044111401$
$b_5 = -0.10714285714285714$	$\hat{b}_5 = 0.25$	$m_5 = 1.0$	$\hat{m}_5 = 1.0$
$b_6 = 0.25$	$\hat{b}_6 = 0.0$	$m_6 = 1.0$	$\hat{m}_6 = 0.0$

Table B.11: Coefficients for RODASP

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
 & 12/23 & 0 & 0 & 0 \\
 & -68/375 & 368/375 & 0 & 0 \\
 & 31/144 & 529/1152 & 125/384 & 0 \\
\hline
 & & & &
\end{array}$$

Table B.12: Butcher array of the four stage CERK method

k	b_{k1}	b_{k2}	b_{k3}	b_{k4}
0	1	0	0	0
1	-65/48	529/384	125/128	-1
2	41/72	-529/576	-125/192	1

Table B.13: Coefficients of the continous extension of the four stage CERK method

Bibliography

[1] Argonne National Laboratory. MPICH, http://www.mcs.anl.gov/research/projects/mpich2/, last accessed on 1/7/2012.

[2] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems. *SIAM J. Num. Anal.*, 39(5):1749–1779, 2002.

[3] K. J. Badcock, I. C. Glover, and B. E. Richards. A Preconditioner for Steady Two-Dimensional Turbulent Flow Simulation. *Int. J. Num. Meth. Heat Fluid Flow*, 6:79–93, 1996.

[4] B. S. Baldwin and H. Lomax. Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows. *AIAA Paper 78-257*, 1978.

[5] D. S. Bale, R. J. LeVeque, S. Mitran, and J. A. Rossmanith. A wave propagation method for conservation laws and balance laws with spatially varying flux functions. *SIAM J. Sci. Comput.*, 24(3):955–978, 2002.

[6] A. L. Banka. Practical Applications of CFD in heat processing. *Heat Treating Progress*, (August), 2005.

[7] T. Barth and M. Ohlberger. Finite Volume Methods: Foundation and Analysis. In E. Stein, R. de Borst, and T. J. R. Hughes, editors, *Encyclopedia of Computational Mechanics*, volume 1: Fundame, chapter 15, pages 439–473. John Wiley and Sons, 2004.

[8] T. J. Barth and D. C. Jesperson. The Design and Application of Upwind Schemes on Unstructured Meshes. *AIAA Paper 89-0366*, 89–0366, 1989.

[9] F. Bassi, A. Ghidoni, and S. Rebay. Optimal Runge-Kutta smoothers for the p-multigrid discontinuous Galerkin solution of the 1D Euler equations. *J. Comp. Phys.*, 11:4153–4175, 2011.

[10] F. Bassi, A. Ghidoni, S. Rebay, and P. Tesini. High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations. *Int. J. Num. Meth. Fluids*, 60:847–865, 2009.

[11] F. Bassi and S. Rebay. Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes equations. *Int. J. Num. Fluids*, 40:197–207, 2002.

[12] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. *Acta Numerica*, 10:1–102, 2001.

[13] O. O. Bendiksen. A new approach to computational aeroelasticity. In *AIAA Paper AIAA-91-0939-CP*, volume AIAA-91-09, pages 1712–1727, 1991.

[14] M. Benzi and D. Bertaccini. Approximate inverse preconditioning for shifted linear systems. *BIT*, 43:231–244, 2003.

[15] M. Benzi and M. Tuma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21:1851–1868, 2000.

[16] H. Bijl and M. H. Carpenter. Iterative solution techniques for unsteady flow computations using higher order time integration schemes. *Int. J. Num. Meth. in Fluids*, 47:857–862, March 2005.

[17] H. Bijl, M. H. Carpenter, V. N. Vatsa, and C. A. Kennedy. Implicit Time Integration Schemes for the Unsteady Compressible Navier-Stokes Equations: Laminar Flow. *J. Comp. Phys.*, 179:313–329, 2002.

[18] P. Birken. Preconditioning GMRES for Steady Compressible Inviscid Flows. Technical report, RWTH-Aachen, IGPM, 2002.

[19] P. Birken. Numerical simulation of tunnel fires using preconditioned finite volume schemes. *ZAMP*, 59:416–433, 2008.

[20] P. Birken. Optimizing Runge-Kutta smoothers for unsteady flow problems. *ETNA*, 39:298–312, 2012.

[21] P. Birken, J. Duintjer Tebbens, A. Meister, and M. Tuma. Updating preconditioners for permuted non-symmetric linear systems. *PAMM*, 7(1):1022101–1022102, 2007.

[22] P. Birken, J. Duintjer Tebbens, A. Meister, and M. Tuma. Preconditioner Updates applied to CFD model problems. *Appl. Num. Math.*, 58:1628–1641, 2008.

[23] P. Birken, G. Gassner, M. Haas, and C.-D. Munz. Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navier-Stokes equations. *J. Comp. Phys.*

[24] P. Birken, G. Gassner, M. Haas, and C.-D. Munz. Efficient Time Integration for Discontinuous Galerkin Methods for the Unsteady 3D Navier-Stokes Equations. In J. Eberhardsteiner, editor, *European Congress on Computational Methods and Applied Sciences and Engineering (ECCOMAS 2012)*, number Eccomas, 2012.

[25] P. Birken and A. Jameson. Nonlinear iterative solvers for unsteady Navier-Stokes equations. In *Proceedings of Hyp2008 - the twelfth International Conference on Hyperbolic Problems*, pages 429–438. AMS, 2009.

[26] P. Birken and A. Jameson. On Nonlinear Preconditioners in Newton-Krylov-Methods for Unsteady Flows. *Int. J. Num. Meth. Fluids*, 62:565–573, 2010.

[27] P. Birken and A. Meister. Stability of Preconditioned Finite Volume Schemes at Low Mach Numbers. *BIT*, 45(3), 2005.

[28] P. Birken, K. J. Quint, S. Hartmann, and A. Meister. A Time-Adaptive Fluid-Structure Interaction Method for Thermal Coupling. *Comp. Vis. in Science*, 13(7):331–340, 2011.

[29] C. Bischof and A. Carle. ADIFOR, http://www.mcs.anl.gov/research/projects/adifor/, last accessed 1/21/2012.

[30] S. Blanes and F. Casas. On the necessity of negative coefficients for operator splitting schemes of order higher than two. *Appl. Num. Math.*, 54(1):23–37, 2005.

[31] J. Blazek. *Computational Fluid Dynamics.* Elsevier, 2nd edition, 2004.

[32] K. E. Brenan, S. L. Campbell, and L. R. Petzold. Numerical solution of initial-value problems in DAEs. *Classicas in Applied Mathematics*, 14, 1996.

[33] J. M. Buchlin. Convective Heat Transfer and Infrared Thermography. *J. Appl. Fluid Mech.*, 3(1):55–62, 2010.

[34] D. A. Caughey and A. Jameson. How Many Steps are Required to Solve the Euler Equations of Steady Compressible Flow: In Search of a Fast Solution Algorithm. *AIAA Paper 2001-2673*, 2001.

[35] CENTAUR Software. CENTAUR Grid Generator, http://www.centaursoft.com/grid-generator, last accessed on 1/20/2012.

[36] G.-Q. Chen and D. Wang. The Cauchy problem for the Euler equations for compressible fluids. In *Handbok on Mathematical Fluid Dynamics*, volume 1. Elsevier, 2002.

[37] A. J. Chorin and J. Marsden. *A Mathematical Introduction to Fluid Mechanics.* Springer Verlag, New York, 1997.

[38] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comp. Appl. Math.*, 86:387–414, 1997.

[39] B. Cockburn and C.-W. Shu. The local discontinuous Galerkin method for time-dependent convection diffusion systems. *SIAM J. Num. Analysis*, 35:2440–2463, 1998.

[40] Tim Colonius and Sanjiva K. Lele. Computational aeroacoustics: progress on nonlinear problems of sound generation. *Progress in Aerospace Sciences*, 40(6):345–416, August 2004.

[41] R. Courant, K. O. Friedrichs, and H. Lewy. Über die partiellen Differentialgleichungen der mathematischen Physik. *Math. Annalen*, 100:32–74, 1928.

[42] M. Crandall and A. Majda. The method of fractional steps for conservation laws. *Numer. Math.*, 34:285–314, 1980.

[43] G. A. Davis and O. O. Bendiksen. Transonic Panel Flutter. In *AIAA 93-1476*, 1993.

[44] T. Davis. UMFPACK. http://www.cise.ufl.edu/research/sparse/umfpack/, last accessed 1/21/2012.

[45] F. Davoudzadeh, H. Mcdonald, and B. E. Thompson. Accuracy evaluation of unsteady CFD numerical schemes by vortex preservation. *Computers & Fluids*, 24:883–895, 1995.

[46] B. de St. Venant. Memoire sur la dynamique des fluides. *C. R. Acad. Sci. Paris*, 17:1240–1242, 1845.

[47] K. Dekker and J. G. Verwer. *Stability of Runge-Kutta methods for stiff nonlinear differential equations*. CWI Monogr. 2. North Holland, Amsterdam, 1984.

[48] S. Dellacherie. Analysis of Godunov type schemes applied to the compressible Euler system at low Mach number. *J. Comp. Phys.*, 229(4):978–1016, February 2010.

[49] R. Dembo, R. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[50] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. SIAM, Philadelphia, 1996.

[51] P. Deuflhard. *Newton Methods*. Springer, 2004.

[52] J. Duintjer Tebbens and M. Tuma. Efficient preconditioning of sequences of nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 2007.

[53] J. Duintjer Tebbens and M. Tuma. Preconditioner updates for solving sequences of linear systems in matrix-free environment. *Num. Lin. Algebra with Appl.*, 17:997–1019, 2010.

[54] R. P. Dwight. *Efficiency Improvements of RANS-based Analysis and Optimization using Implicit and Adjoint Methods on Unstructured Grids*. Phd thesis, University of Manchester, 2006.

[55] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Num. Anal.*, 20(2):345–357, 1983.

[56] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact newton method. *SIAM J. Sci. Comput.*, 17(1):16–32, 1996.

[57] P. Ellsiepen. *Zeits- und ortsadaptive Verfahren angewandt auf Mehrphasenprobleme poröser Medien.* Dissertation, University of Stuttgart, Institute of Mechanics II, 1999.

[58] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21(2):352–362, 1984.

[59] C. Farhat. CFD-based Nonlinear Computational Aeroelasticity. In E. Stein, R. de Borst, and T. J. R. Hughes, editors, *Encyclopedia of Computational Mechanics*, volume 3: Fluids, chapter 13, pages 459–480. John Wiley & Sons, 2004.

[60] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal. p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comp. Phys.*, 207:92–113, 2005.

[61] K. J. Fidkowski and P. L. Roe. An Entropy Adjoint Approach to Mesh Refinement. *SIAM J. Sci. Comput.*, 32(3):1261–1287, 2010.

[62] G. Gassner, M. Dumbser, F. Hindenlang, and C.-D. Munz. Explicit one-step time discretizations for discontinuous Galerkin and finite volume schemes based on local predictors. *J. Comp. Phys.*, 230(11):4232–4247, 2011.

[63] G. Gassner, F. Lörcher, and C.-D. Munz. A contribution to the construction of diffusion fluxes for finite volume and discontinuous Galerkin schemes. *J. Comp. Phys.*, 224:1049–1063, 2007.

[64] G. Gassner, F. Lörcher, and C.-D. Munz. A Discontinuous Galerkin Scheme based on a Space-Time Expansion II. Viscous Flow Equations in Multi Dimensions. *J. Sci. Comput.*, 34:260–286, 2008.

[65] G. J. Gassner, F. Lörcher, C.-D. Munz, and J. S. Hesthaven. Polymorphic nodal elements and their application in discontinuous Galerkin methods. *J. Comp. Phys.*, 228(5):1573–1590, 2009.

[66] T. Gerhold, O. Friedrich, J. Evans, and M. Galle. Calculation of Complex Three-Dimensional Configurations Employing the DLR-TAU-Code. *AIAA Paper*, 97-0167, 1997.

[67] C. Geuzaine and J.-F. Remacle. Gmsh, http://www.geuz.org/gmsh/, last accessed on 1/20/2012.

[68] M. B. Giles. Stability Analysis of Numerical Interface Conditions in Fluid-Structure Thermal Analysis. *Int. J. Num. Meth. in Fluids*, 25:421–436, 1997.

[69] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*, volume 118 of *Applied Mathematical Sciences.* Springer, New York, Berlin, Heidelberg, 1996.

[70] S. K. Godunov. A Finite Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics. *Mat. Sb.*, 47:357–393, 1959.

[71] J. B. Goodman and R. J. LeVeque. On the Accuracy of Stable Schemes for 2D Scalar Conservation Laws. *Math. Comp.*, 45(171):15–21, 1985.

[72] S. Gottlieb, D. I. Ketcheson, and C.-W. Shu. High Order Strong Stability Preserving Time Discretizations. *J. Sci. Comp.*, 38:251–289, 2009.

[73] S. Gottlieb, C.-W. Shu, and E. Tadmor. Strong Stability-Preserving High-Order Time Discretization Methods. *SIAM Review*, 43(1):89–112, 2001.

[74] A. Greenbaum, V. Pták, and Z. Strakoš. Any Nonincreasing Convergence Curve is Possible for GMRES. *SIAM J. Matrix Anal. Appl.*, 17(3):465–469, 1996.

[75] J. J. Greenberg and A. Y. Leroux. A Well-Balanced Scheme for the Numerical Processing of Source Terms in Hyperbolic Equations. *SIAM J. Numer. Anal.*, 33(1):1–16, 1996.

[76] M. J. Grote and T. Huckle. Parallel Preconditioning with Sparse Approximate Inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, 1997.

[77] H. Guillard and C. Farhat. On the significance of the geometric conservation law for flow computations on moving meshes. *Comp. Meth. Appl. Mech. Engrg.*, 190:1467–1482, 2000.

[78] H. Guillard and A. Murrone. On the behavior of upwind schemes in the low Mach number limit: II. Godunov type schemes. *Computers & Fluids*, 33:655–675, May 2004.

[79] H. Guillard and C. Viozat. On the Behaviour of Upwind Schemes in the Low Mach Number Limit. *Computers & Fluids*, 28:63–86, 1999.

[80] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4 of *Springer Series in Computational Mathematics.* Springer, Berlin, Heidelberg, New York, Tokio, 1985.

[81] E. Hairer, S. P. Nø rsett, and G. Wanner. *Solving Ordinary Differential Equations I.* Springer, Berlin, Heidelberg, New York, series in edition, 2000.

[82] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II.* Springer, Berlin, series in edition, 2004.

[83] A. Harten, J. M. Hyman, and P. D. Lax. On Finite-Difference Approximations and Entropy Conditions for Shocks. *Comm. Pure Appl. Math.*, XXIX:297–322, 1976.

[84] R. Hartmann. Adaptive discontinuous Galerkin methods with shock-capturing for the compressible Navier-Stokes equations. *Int. J. Num. Meth. Fluids*, 51:1131–1156, 2006.

[85] S. Hartmann. *Finite-Elemente Berechnung inelastischer Kontinua.* PhD thesis, Kassel, 2003.

[86] S. Hartmann. TASA-FEM: Ein Finite-Elemente-Programm für raum-zeitadaptive gekoppelte Strukturberechnungen. *Mitteilungen des Instituts für Mechanik*, 1, 2006.

[87] S. Hartmann, J. Duintjer Tebbens, K. J. Quint, and A. Meister. Iterative solvers within sequences of large linear systems in non-linear structural mechanics. *ZAMM*, 89(9):711–728, 2009.

[88] P. Haupt. *Continuum Mechanics and Theory of Materials.* Springer, Berlin, 2000.

[89] U. Heck, U. Fritsching, and K. Bauckhage. Fluid flow and heat transfer in gas jet quenching of a cylinder. *International Journal of Numerical Methods for Heat & Fluid Flow*, 11:36–49, 2001.

[90] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications.* Springer, 2008.

[91] M. Hinderks and R. Radespiel. Investigation of Hypersonic Gap Flow of a Reentry Nosecap with Consideration of Fluid Structure Interaction. *AIAA Paper*, 06-1111, 2006.

[92] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, 2005.

[93] C. Hirsch. *Numerical computation of internal and external flows*, volume 1. Wiley & Sons, Chicester, New York, 1988.

[94] C. Hirsch. *Numerical computation of internal and external flows*, volume 2. Wiley & Sons, Chicester, New York, 1988.

[95] K. H. Huebner, D. L. Dewhirst, D. E. Smith, and T. G. Byrom. *The Finite Element Methods for Engineers.* John Wiley & Sons, 4th edition, 2001.

[96] H. T. Huynh. A Flux Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin Methods. In *AIAA Paper AIAA 2007-4079*, 2007.

[97] A. Jameson. Transonic flow calculations for aircraft. In F. Brezzi, editor, *Numerical Methods in Fluid Dynamics*, Lecture Notes in Mathematics, pages 156–242. Springer, 1985.

[98] A. Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. *AIAA Paper 91-1596*, 1991.

[99] A. Jameson. Aerodynamics. In E. Stein, R. de Borst, and T. J. R. Hughes, editors, *Encyclopedia of Computational Mechanics*, volume 3: Fluids, chapter 11, pages 325–406. John Wiley & Sons, 2004.

[100] A. Jameson and T. J. Baker. Solution of the Euler equations for complex configurations. In *Proceedings of AlAA 6th Computational Fluid Dynamics Conference, Danvers*, pages 293–302, 1983.

[101] J. Jeong and F. Hussain. On the identification of a vortex. *J. Fluid Mech.*, 285:69–94, 1995.

[102] V. John and J. Rang. Adaptive time step control for the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engrg.*, 199:514–524, 2010.

[103] G. Jothiprasad, D. J. Mavriplis, and D. A. Caughey. Higher-order time integration schemes for the unsteady Navier-Stokes equations on unstructured meshes. *J. Comp. Phys.*, 191:542–566, 2003.

[104] A. Kanevsky, M. H. Carpenter, D. Gottlieb, and J. S. Hesthaven. Application of implicit-explicit high order Runge-Kutta methods to discontinuous-Galerkin schemes. *J. Comp. Phys.*, 225:1753–1781, 2007.

[105] R. Kannan and Z. J. Wang. A Study of Viscous Flux Formulations for a p-Multigrid Spectral Volume Navier Stokes Solver. *J. Sci. Comput.*, 41(2):165–199, January 2009.

[106] G. E. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, 2nd edition, 2005.

[107] Karypis Lab. ParMETIS, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview, last accessed on 1/7/12.

[108] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia, PA, 1995.

[109] C. A. Kennedy and M. H. Carpenter. Additive Runge-Kutta schemes for convection-diffusion-reaction equations. *Appl. Num. Math.*, 44:139–181, 2003.

[110] D. I. Ketcheson, C. B. Macdonald, and S. Gottlieb. Optimal implicit strong stability preserving Runge-Kutta methods. *Appl. Num. Math.*, 59:373–392, 2009.

[111] C. M. Klaij, J. J. W. van der Vegt, and H. van der Ven. Space-time discontinuous Galerkin method for the compressible Navier-Stokes equations. *J. Comp. Phys.*, 217(2):589–611, 2006.

[112] C. M. Klaij, M. H. van Raalte, J. J. W. van der Vegt, and H. van der Ven. h-Multigrid for space-time discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comp. Phys.*, 227:1024–1045, 2007.

[113] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *J. Comp. Phys.*, 193:357–397, 2004.

[114] D. A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations*. Scientific Computation. Springer, 2000.

[115] D. A. Kopriva, S. L. Woodruff, and M. Y. Hussaini. Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method. *Int. J. Num. Meth. in Eng.*, 53:105–122, 2002.

[116] H.-O. Kreiss and J. Lorenz. *Initial Boundary Value Problems and the Navier-Stokes Equations*. Academic Press, New York, 1989.

[117] D. Kröner and M. Ohlberger. A posteriori error estimates for upwind finite volume schemes for nonlinear conservation laws in multi dimensions. *Math. Comp.*, 69(229):25–39, 1999.

[118] U. Küttler and W. A. Wall. Fixed-point fluidstructure interaction solvers with dynamic relaxation. *Comput. Mech.*, 43:61–72, 2008.

[119] S. Langer. Investigation and application of point implicit RungeKutta methods to inviscid flow problems. *Int. J. Num. Meth. Fluids*, 2011.

[120] S. Langer and D. Li. Application of point implicit RungeKutta methods to inviscid and laminar flow problems using AUSM and AUSM + upwinding. *International Journal of Computational Fluid Dynamics*, 25(5):255–269, 2011.

[121] J. O. Langseth, A. Tveito, and R. Winther. On the Convergence of Operator Splitting applied to Conversation Laws with Source Terms. *SIAM J. Num. Anal.*, 33(3):843–863, 1996.

[122] P. Le Tallec and J. Mouro. Fluid structure interaction with large structural displacements. *Comp. Meth. Appl. Mech. Engrg.*, 190:3039–3067, 2001.

[123] R. J. LeVeque. Hyperbolic Conservation Laws and Numerical Methods. High Resolution (Upwind and TVD) Methods for the Compressible Flow Equations, Selected Special Topics, Von Karman Institute for Fluid Dynamics, Rhode-Saint-Genèse, 1994.

[124] R. J. LeVeque. *Finite Volume methods for Hyperbolic Problems.* Cambridge University Press, Cambridge, 2002.

[125] P.-L. Lions. *Mathematical Topics in Fluid Mechanics Volume 2 Compressible Models.* Oxford Science Publications, 1998.

[126] N. Lior. The cooling process in gas quenching. *J. Materials Processing Technology*, 155-156:1881–1888, 2004.

[127] M.-S. Liou. A sequel to AUSM, Part II: AUSM+-up for all speeds. *J. Comp. Phys.*, 214:137–170, 2006.

[128] F. Lörcher, G. Gassner, and C.-D. Munz. A Discontinuous Galerkin Scheme Based on a Space-Time Expansion. I. Inviscid Compressible Flow in One Space Dimension. *Journal of Scientific Computing*, 32(2):175–199, 2007.

[129] H. Luo, J. D. Baum, and R. Löhner. A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. *J. Comp. Phys.*, 211:767–783, 2006.

[130] K. Mani and D. J. Mavriplis. Efficient Solutions of the Euler Equations in a Time-Adaptive Space-Time Framework. *AIAA-Paper 2011-774*, 2011.

[131] M. F. Maritz and S. W. Schoombie. Exact analysis of nonlinear instability in a discrete Burgers' equation. *J. Comp. Phys.*, 97(1):73–90, 1991.

[132] R. Massjung. *Numerical Schemes and Well-Posedness in Nonlinear Aeroelasticity.* PhD thesis, RWTH Aachen, 2002.

[133] R. Massjung. Discrete conservation and coupling strategies in nonlinear aeroelasticity. *Comp. Meth. Appl. Mech. Engrg.*, 196:91–102, 2006.

[134] H. G. Matthies, R. Niekamp, and J. Steindorf. Algorithms for strong coupling procedures. *Comput. Methods Appl. Mech. Engrg.*, 195:2028–2049, 2006.

[135] H. G. Matthies and J. Steindorf. Algorithms for strong coupling procedures. *Comput. Methods Appl. Mech. Engrg.*, 195:2028–2049, 2003.

[136] D. J. Mavriplis. An Assessment of Linear Versus Nonlinear Multigrid Methods for Unstructured Mesh Solvers. *J. Comp. Phys.*, 175:302–325, 2002.

[137] G. May, F. Iacono, and A. Jameson. A hybrid multilevel method for high-order discretization of the Euler equations on unstructured meshes. *J. Comp. Phys.*, 229:3938–3956, 2010.

[138] P. R. McHugh and D. A. Knoll. Comparison of standard and matrix-free implementations of several Newton-Krylov solvers. *AIAA J.*, 32(12):2394–2400, 1994.

[139] R. C. Mehta. Numerical Computation of Heat Transfer on Reentry Capsules at Mach 5. *AIAA-Paper 2005-178*, 2005.

[140] A. Meister. *Zur zeitgenauen numerischen Simulation reibungsbehafteter, kompressibler, turbulenter Strömungsfelder mit einer impliziten Finite-Volumen-Methode vom Box-Typ.* Dissertation, Technische Hochschule Darmstadt, 1996.

[141] A. Meister. *Numerik linearer Gleichungssysteme, Eine Einführung in moderne Verfahren.* Vieweg, Wiesbaden, 1999.

[142] A. Meister and Th. Sonar. Finite-volume schemes for compressible fluid flow. *Surv. Math. Ind.*, 8:1–36, 1998.

[143] A. Meister and C. Vömel. Efficient Preconditioning of Linear Systems arising from the Discretization of Hyperbolic Conservation Laws. *Adv. in Comput. Math.*, 14:49–73, 2001.

[144] C. Michler, E. H. van Brummelen, and R. de Borst. Error-amplification Analysis of Subiteration-Preconditioned GMRES for Fluid-Structure Interaction. *Comp. Meth. Appl. Mech. Eng.*, 195:2124–2148, 2006.

[145] P. Moin and K. Mahesh. DIRECT NUMERICAL SIMULATION: A Tool in Turbulence Research. *Ann. Rev. Fluid Mech.*, 30:539–578, 1998.

[146] K. W. Morton and Th. Sonar. Finite volume methods for hyperbolic conservation laws. *Acta Numerica*, pages 155–238, 2007.

[147] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13(3):778–795, 1992.

[148] C. R. Nastase and D. J. Mavriplis. High-order discontinuous Galerkin methods using an hp-multigrid approach. *J. Comp. Phys.*, 213:330–357, 2006.

[149] C. L. M. H. Navier. Memoire sur les lois du mouvement des fluides. *Mem. Acad. R. Sci. Paris*, 6:389–416, 1823.

[150] M. Ohlberger. A posteriori error estimate for finite volume approximations to singularly perturbed nonlinear convection-diffusion equations. *Numer. Math.*, 87:737–761, 2001.

[151] H. Olsson and G. Söderlind. The Approximate Runge-Kutta Computational Process. *BIT*, 40:351–373, 2000.

[152] S. Osher and S. Chakravarthy. High resolution schemes and the entropy condition. *SIAM J. Num. Anal.*, 21(5):955–984, 1984.

[153] B. Owren and M. Zennaro. Order Barriers for Continuous Explicit Runge-Kutta methods. *Math. Comp.*, 56(194):645–661, 1991.

[154] H. Park, R. R. Nourgaliev, R. C. Martineau, and D. A. Knoll. On physics-based preconditioning of the NavierStokes equations. *J. Comp. Phys.*, 228(24):9131–9146, 2009.

[155] V. C. Patel, W. Rodi, and G. Scheuerer. Turbulence Models for Near-Wall and Low-Reynolds Number Flows: A Review. *AIAA Journal*, 23(9):1308–1319, 1985.

[156] J. Peraire and P.-O. Persson. The Compact Discontinuous Galerkin (CDG) Method for Elliptic Problems. *SIAM J. Sci. Comput.*, 30(4):1806–1824, 2008.

[157] P.-O. Persson and J. Peraire. Newton-GMRES Preconditioning for Discontinuous Galerkin discretizations of the Navier-Stokes equations. *SIAM J. Sci. Comp.*, 30:2709–2733, 2008.

[158] S. D. Poisson. Memoire sue les equations generales de l'equilibre et du mouvement des corps solides elastiques et des fluides. *J. de l'Ecole Polytechnique de Paris*, 13:139–166, 1831.

[159] N. Qin, D. K. Ludlow, and S. T. Shaw. A matrix-free preconditioned Newton/GMRES method for unsteady Navier-Stokes solutions. *Int. J. Num. Meth. Fluids*, 33:223–248, 2000.

[160] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, Oxford, 1999.

[161] J. Rang and L. Angermann. New Rosenbrock W-Methods of Order 3 for Partial Differential Algebraic Equations of Index 1. *BIT*, 45:761–787, 2005.

[162] J. Reisner, V. Mousseau, A. Wyszogrodzki, and D. A. Knoll. A fully implicit hurricane model with physics-based preconditioning. *Monthly Weather Review*, 133:1003–1022, 2005.

[163] J. Reisner, A. Wyszogrodzki, V. Mousseau, and D. Knoll. An efficient physics-based preconditioner for the fully implicit solution of small-scale thermally driven atmospheric flows. *J. Comp. Phys.*, 189(1):30–44, 2003.

[164] F. Reitsma, G. Strydom, J. B. M. de Haas, K. Ivanov, B. Tyobeka, R. Mphahlele, T. J. Downar, V. Seker, H. D. Gougar, D. F. Da Cruz, and U. E. Sikik. The PBMR steadystate and coupled kinetics core thermal-hydraulics benchmark test problems. *Nuclear Engineering and Design*, 236(5-6):657–668, 2006.

[165] W. C. Rheinboldt. *Methods for Solving Systems of Nonlinear Equations.* SIAM, 2nd edition, 1998.

[166] F. Rieper. A low-Mach number fix for Roe's approximate Riemann solver. *J. Comp. Phys.*, 2011.

[167] F. Rieper and G. Bader. Influence of cell geometry on the accuracy of upwind schemes in the low Mach number regime. *J. Comp. Phys.*, 228:2918–2933, 2009.

[168] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.

[169] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston, 1996.

[170] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[171] B. Saba and K. Steinhoff. Massivumformprodukte mit funktional gradierten Eigenschaften durch eine differenzielle thermo-mechanische Prozessführung. *WT-Online*, pages 745–752, 2007.

[172] H. Schlichting and K. Gersten. *Boundary Layer Theory.* Springer, 8th revise edition, 2000.

[173] S. Schüttenberg, M. Hunkel, U. Fritsching, and H.-W. Zoch. Controlling of Distortion by means of Quenching in adapted Jet Fields. *Materialwissenschaft und Werkstofftechnik*, 37(1):92–96, 2006.

[174] L. F. Shampine. *Numerical solution of ordinary differential equations.* Springer, 1994.

[175] C.-W. Shu. Total-variation diminishing time discretizations. *SIAM J. Sci. Stat. Comp.*, 9:1073–1084, 1988.

[176] C.-W. Shu and S. Osher. Efficient Implementation of Essentially Nonoscillatory Shock-Capturing Schemes. *J. Comp. Phys.*, 77:439–471, 1988.

[177] J. Smagorinsky. General Circulation Experiments with the Primitive Equations. *Monthly Weather Review*, 91:99–165, 1963.

[178] G. Söderlind. Digital Filters in Adaptive Time-Stepping. *ACM TOMS*, 29(1):1–26, 2003.

[179] G. Söderlind. Time-step selection algorithms: Adaptivity, control, and signal processing. *Appl. Num. Math.*, 56:488–502, 2006.

[180] G. Söderlind and L. Wang. Adaptive time-stepping and computational stability. *J. Comp. Appl. Math.*, 185:225 – 243, 2006.

[181] G. Söderlind and L. Wang. Evaluating numerical ODE/DAE methods, algorithms and software. *J. Comp. Appl. Math.*, 185:244 – 260, 2006.

[182] P. Sonneveld and M. B. van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM J. Sc. Comp.*, 31(2):1035–1062, 2008.

[183] P. R. Spalart and S. R. Allmaras. A One-Equation Turbulence Model for Aerodynamic Flows. *AIAA 30th Aerospace Science Meeting*, 92–0439, 1992.

[184] P. R. Spalart, W. Jou, M. Strelets, and S. R. Allmaras. Comments on the Feasibility of LES for Wings, and on a Hybrid RANS/LES Approach. In *1st ASOSR CONFERENCE on DNS/LES. Arlington, TX*, 1997.

[185] S. P. Spekreijse. Multigrid Solution of Monotone Second-Order Discretizations of Hyperbolic Conservation Laws. *Math. Comp.*, 49(179):135–155, 1987.

[186] A. St-cyr and D. Neckels. A Fully Implicit Jacobian-Free High-Order Discontinuous Galerkin Mesoscale Flow Solver. In *Proceedings of the 9th International Conference on Computational Science*, pages 243–252, Berlin, Heidelberg, 2009. Springer.

[187] G. G. Stokes. On the theories of the internal friction of fluids in motion. *Trans. Camb. Phil. Soc.*, 8:287–305, 1845.

[188] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Num. Anal.*, 5(3):506–517, 1968.

[189] P. Stratton, I. Shedletsky, and M. Lee. Gas Quenching with Helium. *Solid State Phenomena*, 118:221–226, 2006.

[190] K. Strehmel and R. Weiner. *Linear-implizite Runge-Kutta-Methoden und ihre Anwendung.* Teubner, Stuttgart, 1992.

[191] M. Svärd, M. H. Carpenter, and J. Nordström. A stable high-order finite difference scheme for the compressible Navier-Stokes equations, far-field boundary conditions. *J. Comp. Phys.*, 225(1):1020–1038, 2007.

[192] M. Svärd and J. Nordström. Well-Posed Boundary Conditions for the Navier-Stokes Equations. *SIAM J. Num. Anal.*, 30(3):797, 2005.

[193] T. Tang and Z.-H. Teng. Error bounds for fractional step methods for conservation laws with source terms. *SIAM J. Num. Anal.*, 32(1):110–127, 1995.

[194] Open MPI Development Team. OpenMPI.

[195] K. Thompson. Time dependent boundary conditions for hyperbolic systems. *J. Comp. Phys.*, 68:1–24, 1987.

[196] B. Thornber, A. Mosedale, D. Drikakis, and D. Youngs. An improved reconstruction method for compressible flows with low Mach number features. *J. Comp. Phys.*, 227:4873–4894, 2008.

[197] V. A. Titarev and E. F. Toro. ADER: Arbitrary High Order Godunov Approach. *J. Sci. Comp.*, 17(1-4):609–618, 2002.

[198] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, Berlin, Heidelberg, New York, 2. edition, 1999.

[199] L. N. Trefethen. Pseudospectra of Linear Operators. *SIAM Review*, 39 (3):383–406, 1997.

[200] D. Tromeur-Dervout and Y. Vassilevski. Choice of initial guess in iterative solution of series of systems arising in fluid flow simulations. *J. Comp. Phys.*, 219:210–227, 2006.

[201] U. Trottenberg, C. W. Oosterlee, and S. Schüller. *Multigrid*. Elsevier Academic Press, 2001.

[202] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.

[203] H. A. van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[204] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*, volume 13 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

[205] H. A. van der Vorst and C. Vuik. GMRESR: a Family of Nested GMRES methods. *Num. Lin. Algebra with Appl.*, 1(4):369–386, 1994.

[206] E. R. Van Driest. National Advisory Commitee for Aeronautics (NACA) - Investigation of laminar boundary layer in compressible fluids using the crocco method. *NACA*, 1952.

[207] B. van Leer. Flux-vector splitting for the Euler equations. In E Krause, editor, *Eighth International Conference on Numerical Methods in Fluid Dynamics*, number 170 in Lecture Notes in Physics, pages 507–512, Berlin, 1982. Springer Verlag.

[208] B. van Leer, C.-H. Tai, and K. G. Powell. Design of Optimally Smoothing Multi-Stage Schemes for the Euler Equations. In *AIAA 89-1933-CP*, pages 40–59, 1989.

[209] A. van Zuijlen and H. Bijl. Implicit and explicit higher order time integration schemes for structural dynamics and fluid-structure interaction computations. *Comp. & Struct.*, 83:93–105, 2005.

[210] A. van Zuijlen, A. de Boer, and H. Bijl. Higher-order time integration through smooth mesh deformation for 3D fluidstructure interaction simulations. *J. Comp. Phys*, 224:414–430, 2007.

[211] R. S. Varga. *Matrix Iterative Analysis*, volume 27 of *Series in Computational Mathematics*. Springer, New York, Berlin, Heidelberg, 2000.

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

[213] J. Vierendeels, L. Lanoye, J. Degroote, and P. Verdonck. Implicit Coupling of Partitioned Fluid-Structure Interaction Problems with Reduced Order Models. *Comp. & Struct.*, 85:970–976, 2007.

[214] P. Vijayan and Y. Kalinderis. A 3D Finite-Volume Scheme for the Euler Equations on Adaptive Tetrahedral Grids. *J. Comp. Phys.*, 113:249–267, 1994.

[215] P. E. Vincent, P. Castonguay, and A. Jameson. A New Class of High-Order Energy Stable Flux Reconstruction Schemes. *J. Sci. Comp.*, 47:50–72, 2011.

[216] P. E. Vincent and A. Jameson. Facilitating the Adoption of Unstructured High-Order Methods Amongst a Wider Community of Fluid Dynamicists. *Math. Model. Nat. Phenom.*, 6(3):97–140, 2011.

[217] Y. Wada and M.-S. Liou. A Flux Splitting Scheme with High-Resolution and Robustness for Discontinuities. *AIAA Paper 94-0083*, 94–0083, 1994.

[218] K. Wang, W. Xue, H. Lin, S. Xu, and W. Zheng. Updating preconditioner for iterative method in time domain simulation of power systems. *Science China Technological Sciences*, 54(4):1024–1034, February 2011.

[219] L. Wang and D. J. Mavriplis. Implicit solution of the unsteady Euler equations for high-order accurate discontinuous Galerkin discretizations. *J. Comp. Phys.*, 225:1994–2015, 2007.

[220] R. Weiner, B. A. Schmitt, and H. Podhaisky. ROWMAP a ROW-code with Krylov techniques for large stiff ODEs. *Appl. Num. Math.*, 25:303–319, 1997.

[221] P. Wesseling. *Principles of Computational Fluid Dynamics.* Springer, 2001.

[222] P. Wesseling. *An Introduction to Multigrid Methods.* R T Edwards Inc, 2004.

[223] L. B. Wigton, N. J. Yu, and D. P. Young. GMRES acceleration of Computational Fluid Dynamics Codes. *AIAA Paper 85-1494*, A85-40933, 1985.

[224] P. W. Yarrington and E. A. Thornton. Finite Element Analysis of Low-Speed Compressible Flows Within Convectively Cooled Structures. *J. Thermophysics and Heat Transfer*, 8(4):678–686, 1994.

[225] F. Zahle, N. N. Soerensen, and J. Johansen. Wind Turbine Rotor-Tower Interaction Using an Incompressible Overset Grid Method. *Wind Energy*, 12:594–619, 2009.

[226] Q. Zhang and C.-W. Shu. Error Estimates to Smooth Solutions of Runge-Kutta Discontinuous Galerkin Method for Symmetrizable Systems of Conservation Laws. *SIAM J. Num. Anal.*, 44(4):1703–1720, 2006.