Nečas Center for Mathematical Modeling

A parallel fully-implicit discontinuous Galerkin method for the compressible flows

V. Dolejší, V. Šíp

Preprint no. 2013-016

http://ncmm.karlin.mff.cuni.cz/

A parallel fully-implicit discontinuous Galerkin method for the compressible flows

Vít Dolejší, Viktor Šíp

May 3, 2013

Abstract

We deal with the numerical simulation of a motion of inviscid as well as viscous compressible fluids. We discretize the governing Navier-Stokes and the Euler equations by the backward difference formula – discontinuous Galerkin finite element (BDF-DGFE) method, which exhibits a sufficiently stable, efficient and accurate numerical scheme. We focus on the parallelization of this method. We describe the implementation issues and present several numerical experiments demonstrating its efficiency.

Key words: discontinuous Galerkin method, compressible Navier-Stokes equations, linear algebra problems, preconditioning, stopping criterion, choice of the time step
AMS Subject Classification: 76M10, 76N15, 35Q35, 65L06

Contents

1 Introduction

Our aim is to develop a sufficiently robust, efficient and accurate numerical scheme for the simulation of steady as well as unsteady viscous compressible flows. The *discontinuous Galerkin* (DG) methods have become very popular numerical technique for the solution of the compressible Navier-Stokes equations. DG space discretization uses (higher order) piecewise polynomial discontinuous approximation on arbitrary meshes. DG methods were employed in many papers for the discretization of compressible fluid flow problems, see, e.g., [2, 3, 4, 22, 10, 23, 24, 27, 28, 16, 18, 11, 9, 15, 19], and the references cited therein. Recent progress of the use of the DG method for compressible flow simulations can be found in [29].

The time discretization can be carried out also by a discontinuous approximation (e.g., [27, 20]) but the most usual approach is an application of the method of lines. In this case, Runge-Kutta methods are very popular for their simplicity and high order of accuracy, see [2, 4, 6, 10]. Their drawback is a strong restriction of the size of the time step. To avoid this disadvantage, it is suitable to use an implicit time discretization, e.g., [3, 23]. However, a fully implicit scheme leads to the necessity to solve a nonlinear system of algebraic equations at each time level, which is rather expensive. Therefore, in [11, 13], we developed the *semi-implicit method* which is based on a suitable linearization of the inviscid and viscous fluxes. The linear terms are treated implicitly (by a multistep BDF formula), whereas the nonlinear ones by an explicit extrapolation, which leads to a linear algebraic system at each time level. We called this approach the *backward difference formula – discontinuous Galerkin finite element* (BDF-DGFE) method.

The BDF-DGFE method leads to a sequence of linear algebraic systems which should be solved by a suitable solver. It is advantageous to use an iterative method (e.g., GMRES method [30] with a suitable preconditioner), since the solution of the previous system can be used as an initial guess of the solution of the next system. Moreover, it is not necessary to solve the systems too precisely, since they arise from a discretization of partial differential equations and therefore the systems already contain discretization errors. Numerical experiments presented in [11] showed that the BDF-DGFE method is efficient for unsteady flow problems but its efficiency for steady-state flow regimes is very low. The solution of the linear algebraic systems consumes more than 90% of the total computational time.

Therefore, we develop a new solution strategy which significantly reduces the computational time for steady state flows in comparison with [11]. We were inspired by the idea of the *inexact Newton method* [7], where a sequence of linear algebraic systems also has to be solved. The key is to define a relatively weak stopping criterion which guarantees convergence to the steady state solution but requires only a few GMRES steps at each time level.

Although we focus in this paper on steady-state flows, we employ the "unsteady" formulation since our aim is to solve also unsteady flows with the same method. This approach is practical in situations when it is not known a priori if the flow regime is steady or unsteady. Therefore, we develop a simple technique for the adaptive choice of the size of the time step.

The content of the rest of the paper is the following. In Section 2, we introduce the system of the compressible Navier-Stokes equations. In Section 3, we recall the BDF-DGFE discretization of the Navier-Stokes equations from [11]. In Section 4, we discuss the numerical solution of the arising linear algebraic systems. Particularly, we deal with the choice of preconditioner of the GMRES method, stopping criteria and the size of the time step. Section 5 describe the paralelization of the presented numerical method. Concluding remarks are given in Section 6.

2 Compressible flow problem

Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, be a bounded domain with a piecewise polynomial Lipschitz boundary and $T > 0$. We set $Q_T = \Omega \times (0, T)$ and by $\partial\Omega$ denote the boundary of Ω which consists of

several disjoint parts. We distinguish inlet $\partial\Omega_i$, outlet $\partial\Omega_o$ and impermeable walls $\partial\Omega_W$, i.e. $\partial\Omega = \partial\Omega_i \cup \partial\Omega_o \cup \partial\Omega_W$. The system of the Navier-Stokes equations describing the motion of a non-stationary viscous compressible flow can be written in the dimensionless form

$$\frac{\partial \boldsymbol{w}}{\partial t} + \sum_{s=1}^{d} \frac{\partial \boldsymbol{f}_s(\boldsymbol{w})}{\partial x_s} = \sum_{s=1}^{d} \frac{\partial}{\partial x_s} \left(\sum_{k=1}^{d} \boldsymbol{K}_{sk}(\boldsymbol{w}) \frac{\partial \boldsymbol{w}}{\partial x_k} \right) \quad \text{in } Q_T, \tag{1}$$

where

$$\begin{array}{ll} \boldsymbol{w} = \boldsymbol{w}(x,t) : Q_T \to \mathbb{R}^{d+2}, & \text{the state vector,} \\ \boldsymbol{f}_s : \mathbb{R}^{d+2} \to \mathbb{R}^{d+2},\ s = 1,\dots,d, & \text{the inviscid fluxes,} \\ \boldsymbol{K}_{sk} : \mathbb{R}^{d+2} \to \mathbb{R}^{(d+2)\times(d+2)},\ s,k = 1,\dots,d, & \text{the viscous terms.} \end{array} \tag{2}$$

The forms of vectors $\boldsymbol{w}$, $\boldsymbol{f}_s$, $s = 1,\dots,d$, and matrices $\boldsymbol{K}_{sk}$, $s = 1,\dots,d$, can be found, e.g., in [11] or [17, Section 4.3]. We consider the Newtonian type of fluid accompanied by the state equation of a perfect gas and the definition of total energy. System (1) is of *hyperbolic-parabolic* type and it is equipped with suitable initial and boundary conditions, see [10, 11]. On the inlet and outlet, we prescribe Dirichlet boundary conditions for some of the flow variables, while Neumann conditions are used for the remaining variables. On the impermeable walls, we put

$$\boldsymbol{v} = 0, \qquad \partial\theta/\partial\boldsymbol{n} = 0 \quad \text{ on } \partial\Omega_W, \tag{3}$$

where $\boldsymbol{v}$ is the velocity vector and $\partial\theta/\partial\boldsymbol{n}$ denotes the normal derivative of the temperature.

The problem of solving the Navier-Stokes equations (1) equipped with initial and boundary conditions will be denoted by (CFP) (compressible flow problem).

Let us mention that the Euler fluxes $\boldsymbol{f}_s$, $s = 1,\dots,d$, satisfy (see [17, Lemma 3.1]) $\boldsymbol{f}_s(\boldsymbol{w}) = \boldsymbol{A}_s(\boldsymbol{w})\boldsymbol{w}$, $s = 1,\dots,d$, where $\boldsymbol{A}_s(\boldsymbol{w}) = \frac{D\boldsymbol{f}_s(\boldsymbol{w})}{D\boldsymbol{w}}$, $s = 1,\dots,d$, are the Jacobi matrices of $\boldsymbol{f}_s$. Moreover, we define the matrix

$$\boldsymbol{P}(\boldsymbol{w},\boldsymbol{n}) = \sum_{s=1}^{d} \boldsymbol{A}_s(\boldsymbol{w}) n_s, \tag{4}$$

where $\boldsymbol{n} = (n_1,\dots,n_d) \in \mathbb{R}^d$, $|n|^2 = \sum_{l=1}^{d} n_l^2 = 1$, which plays a role in the definition of the numerical flux. Finally, if $\boldsymbol{w}$ is the state vector satisfying the wall boundary condition (3) then

$$\sum_{k=1}^{d} \boldsymbol{K}_{s,k}(\boldsymbol{w}) \frac{\partial \boldsymbol{w}}{\partial x_k}\bigg|_{\partial\Omega_W} = (0, \boldsymbol{t}_{1s}, \dots, \boldsymbol{t}_{ds}, 0)^{\mathrm{T}} =: \sum_{k=1}^{d} \boldsymbol{K}^W_{s,k}(\boldsymbol{w}) \frac{\partial \boldsymbol{w}}{\partial x_k}\bigg|_{\partial\Omega_W}, \quad s = 1,\dots,d, \tag{5}$$

where $\boldsymbol{t}_{ij}$ are the components of the stress tensor and $\boldsymbol{K}^W_{s,k}$, $s,k = 1,\dots,d$ are matrices which have the last row equal to zeros and the other rows are identical with the rows of $\boldsymbol{K}_{s,k}$, $s,k = 1,\dots,d$.

3 DGFE discretization

3.1 Triangulations

Let $\mathscr{T}_h$ ($h > 0$) be a partition of the closure $\overline{\Omega}$ of the domain Ω into a finite number of closed d-dimensional elements K with mutually disjoint interiors. I.e., $\overline{\Omega} = \bigcup_{K\in\mathscr{T}_h} K$. Moreover, let $F_K : \hat{K} \to \mathbb{R}^d$ be a polynomial mapping such that $F_K(\hat{K}) = K$ where $\hat{K} = \{(\hat{x}_1,\dots,\hat{x}_d);\ \hat{x}_i \geq 0,\ i = 1,\dots,d,\ \sum_{i=1}^{d} \hat{x}_i \leq 1\}$ is the reference simplex.

If $K \cap \partial\Omega = \emptyset$ or $K \cap \partial\Omega$ is a straight line, then F_K is an affine mapping and K is a simplex. Otherwise, F_K is a polynomial mapping of the same degree as the segment $K \cap \partial\Omega$ and K is a curved simplex. We call $\mathscr{T}_h = \{K\}_{K\in\mathscr{T}_h}$ a *triangulation* of Ω and do not require the conforming properties from the finite element method, see, e.g, [5].

By $\mathscr{F}_h$ we denote the set of all open $(d-1)$-dimensional faces (open edges when $d = 2$ or open faces when $d = 3$) of all elements $K \in \mathscr{T}_h$. Further, the symbol $\mathscr{F}_h^I$ stands for the set of all

$\Gamma \in \mathscr{F}_h$ that are contained in Ω (inner faces). Moreover, we define $\mathscr{F}_h^W$, $\mathscr{F}_h^i$ and $\mathscr{F}_h^o$ as the sets of all $\Gamma \in \mathscr{F}_h$ such that $\Gamma \subset \partial\Omega_W$, $\Gamma \subset \partial\Omega_i$ and $\Gamma \subset \partial\Omega_o$, respectively. In order to simplify the notation, we put $\mathscr{F}_h^{io} = \mathscr{F}_h^i \cup \mathscr{F}_h^o$ and $\mathscr{F}_h^B = \mathscr{F}_h^W \cup \mathscr{F}_h^i \cup \mathscr{F}_h^o$.

Finally, for each $\Gamma \in \mathscr{F}_h$ we define a unit normal vector $\boldsymbol{n}_\Gamma$. We assume that for $\Gamma \in \mathscr{F}_h^B$ the vector $\boldsymbol{n}_\Gamma$ has the same orientation as the outer normal of $\partial\Omega$. For each $\Gamma \in \mathscr{F}_h^I$, the orientation of $\boldsymbol{n}_\Gamma$ is arbitrary but fixed.

3.2 Discontinuous finite element spaces

DGFE allows the use of different polynomial degrees over elements. Therefore, although all numerical experiments in this article are carried out for a constant polynomial degree, we assign a positive integer p_K (local polynomial degree) to each $K \in \mathscr{T}_h$. Then we define the vector $p = \{p_K, K \in \mathscr{T}_h\}$. Over the triangulation $\mathscr{T}_h$ we define the finite dimensional space of discontinuous piecewise polynomial functions associated with the vector p by

$$S_h^p = \{v;\ v \in L^2(\Omega),\ v|_K \circ F_K \in P_{p_K}(\hat{K})\ \forall K \in \mathscr{T}_h\}, \tag{6}$$

where $P_{p_K}(\hat{K})$ denotes the space of all polynomials on $\hat{K}$ of degree $\le p_K$, $K \in \mathscr{T}_h$. We seek the approximate solution in the space of vector-valued functions $\boldsymbol{S}_h^p = [S_h^p]^{d+2}$, whose dimension is $N_h = \frac{(d+2)}{d!}\sum_{K\in\mathscr{T}_h}(\Pi_{i=1}^d(p_K+i))$.

For each $\Gamma \in \mathscr{F}_h^I$ there exist two elements $K^{(+)}, K^{(-)} \in \mathscr{T}_h$ such that $\Gamma \subset K^{(+)} \cap K^{(-)}$. We use the convention that $K^{(-)}$ lies in the direction of $\boldsymbol{n}_\Gamma$ and $K^{(+)}$ in the opposite direction of $\boldsymbol{n}_\Gamma$. Then for $v \in S_h^p$, we introduce the notation: $v|_\Gamma^{(+)}$ is the trace of $v|_{K^{(+)}}$ on Γ, $v|_\Gamma^{(-)}$ is the trace of $v|_{K^{(-)}}$ on Γ, and $\langle v\rangle_\Gamma := \left(v|_\Gamma^{(+)} + v|_\Gamma^{(-)}\right)/2$, $[\![v]\!]_\Gamma := v|_\Gamma^{(+)} - v|_\Gamma^{(-)}$. In case that $[\![\cdot]\!]_\Gamma$ and $\langle\cdot\rangle_\Gamma$ are arguments of $\int_\Gamma \ldots \mathrm{d}S$, $\Gamma \in \mathscr{F}_h$ we omit the subscript Γ and write simply $[\![\cdot]\!]$ and $\langle\cdot\rangle$, respectively. Finally, for $\Gamma \in \mathscr{F}_h^B$ we denote by $v|_\Gamma^{(+)}$ the trace of $v|_{K^{(+)}}$ on Γ, where $K^{(+)} \in \mathscr{T}_h$ such that $\Gamma \subset K^{(+)} \cap \partial\Omega$.

3.3 Discretization of the Navier-Stokes equations

In this section, we recall the *backward difference formula – discontinuous Galerkin finite element* (BDF-DGFE) method for the solution of the Navier-Stokes equations (1) presented in [11]. However, in comparison with [11], we employ a little different treatment of the boundary conditions which has better convergence properties.

3.3.1 Inviscid terms

For $\boldsymbol{w}_h, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p$, we define the forms

$$\begin{aligned} \boldsymbol{b}_h(\boldsymbol{w}_h, \boldsymbol{\varphi}_h) \ :=\ & \sum_{\Gamma\in\mathscr{F}_h}\int_\Gamma \left(\boldsymbol{P}^+\left(\langle \boldsymbol{w}_h\rangle, \boldsymbol{n}\right)\boldsymbol{w}_h|_\Gamma^{(+)} + \boldsymbol{P}^-\left(\langle \boldsymbol{w}_h\rangle, \boldsymbol{n}\right)\boldsymbol{w}_h|_\Gamma^{(-)}\right)\cdot[\![\boldsymbol{\varphi}_h]\!]\,\mathrm{d}S \\ & - \sum_{K\in\mathscr{T}_h}\int_K \sum_{s=1}^d \boldsymbol{A}_s(\boldsymbol{w}_h)\boldsymbol{w}_h\cdot\frac{\partial\boldsymbol{\varphi}_h}{\partial x_s}\,\mathrm{d}x, \end{aligned} \tag{7}$$

where $\boldsymbol{A}_s$ are the Jacobi matrices of the inviscid fluxes $\boldsymbol{f}_s$, $s = 1, \ldots, d$, $\boldsymbol{P}^\pm$ are the positive and negative parts of the matrix $\boldsymbol{P}$ given by (4) which define the Vijayasundaram numerical flux [31] used for the approximation of inviscid fluxes through $\Gamma \in \mathscr{F}_h$. For $\Gamma \in \mathscr{F}_h^B$, we have to specify the meaning of $\boldsymbol{w}_h|_\Gamma^{(-)}$. For $\Gamma \in \mathscr{F}_h^W$, we put

$$\boldsymbol{w}_h|_\Gamma^{(-)} := \mathcal{M}\left(\boldsymbol{w}_h|_\Gamma^{(+)}\right), \tag{8}$$

where $\mathcal{M}: \mathbb{R}^{d+2} \to \mathbb{R}^{d+2}$ is the *"mirror operator"* defined on $\Gamma \in \partial\Omega_W$ such that

$$\boldsymbol{w} = (\rho, \boldsymbol{v}, e)^{\mathrm{T}} \;\Rightarrow\; \begin{cases} \mathcal{M}(\boldsymbol{w}) = (\rho, \rho\boldsymbol{v} - 2\rho(\boldsymbol{v}\cdot\boldsymbol{n})\boldsymbol{n}, e)^{\mathrm{T}} & \text{for inviscid flow,} \\ \mathcal{M}(\boldsymbol{w}) = (\rho, -\rho\boldsymbol{v}, e)^{\mathrm{T}} & \text{for viscous flow,} \end{cases} \tag{9}$$

where ρ is the density, e is the total energy, $\boldsymbol{v}$ is the velocity vector and $\boldsymbol{n}$ is the unit outer normal to $\partial\Omega_W$. Therefore, for inviscid flow, the normal components of the velocities of $\boldsymbol{w}$ and $\mathcal{M}(\boldsymbol{w})$ have the same magnitude but opposite direction. For viscous flow, the velocities of $\boldsymbol{w}$ and $\mathcal{M}(\boldsymbol{w})$ have the same magnitude and the opposite directions. The remaining components of $\boldsymbol{w}$ and $\mathcal{M}(\boldsymbol{w})$ (density, energy and tangential component of the velocity) are the same.

Finally, for $\Gamma \in \mathscr{F}_h^{io}$, we put

$$\boldsymbol{w}_h|_\Gamma^{(-)} := LRP(\boldsymbol{w}_h|_\Gamma^{(+)}, \boldsymbol{w}_D, \boldsymbol{n}_\Gamma),\ \Gamma \in \mathscr{F}_h^{io}, \tag{10}$$

where $LRP(\cdot,\cdot,\cdot)$ represents the solution of the *local Riemann problem* considered on edge $\Gamma \in \mathscr{F}_h^{io}$ and $\boldsymbol{w}_D$ is a given state vector (e.g. from far-field boundary conditions), see [12]. For more details, we refer to [13] or [14].

3.3.2 Viscous terms

For $\boldsymbol{w}_h, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p$, we define the forms

$$\begin{aligned} \boldsymbol{a}_h(\boldsymbol{w}_h, \boldsymbol{\varphi}_h) \;=\; & \sum_{K\in\mathscr{T}_h} \int_K \sum_{s,k=1}^{d} \left(\boldsymbol{K}_{s,k}(\boldsymbol{w}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k} \right) \cdot \frac{\partial \boldsymbol{\varphi}_h}{\partial x_s} \,\mathrm{d}x \\ & - \sum_{\Gamma\in\mathscr{F}_h^I} \int_\Gamma \sum_{s=1}^{d} \left\langle \sum_{k=1}^{d} \boldsymbol{K}_{s,k}(\boldsymbol{w}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k} \right\rangle n_s \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S \\ & - \sum_{\Gamma\in\mathscr{F}_h^{io}} \int_\Gamma \sum_{s,k=1}^{d} \boldsymbol{K}_{s,k}(\boldsymbol{w}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k} \, n_s \cdot \boldsymbol{\varphi}_h \,\mathrm{d}S \\ & - \sum_{\Gamma\in\mathscr{F}_h^W} \int_\Gamma \sum_{s,k=1}^{d} \boldsymbol{K}_{s,k}^W(\boldsymbol{w}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k} \, n_s \cdot \boldsymbol{\varphi}_h \,\mathrm{d}S \end{aligned} \tag{11}$$

where $\boldsymbol{K}_{s,k}^W,\ s,k = 1,\dots,d$ are defined by (5).

3.3.3 Interior and boundary penalties

For $\boldsymbol{w}_h, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p$, we define the forms

$$\begin{aligned} \boldsymbol{J}_h^\sigma(\boldsymbol{w}_h, \boldsymbol{\varphi}_h) \;=\; & \sum_{\Gamma\in\mathscr{F}_h^I} \int_\Gamma \sigma [\![\boldsymbol{w}_h]\!] \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S + \sum_{\Gamma\in\mathscr{F}_h^{io}} \int_\Gamma \sigma(\boldsymbol{w}_h - \mathcal{B}(\boldsymbol{w}_h)) \cdot \boldsymbol{\varphi}_h \,\mathrm{d}S \\ & + \sum_{\Gamma\in\mathscr{F}_h^W} \int_\Gamma \sigma(\boldsymbol{w}_h - \mathcal{B}(\boldsymbol{w}_h)) \cdot \mathcal{V}(\boldsymbol{\varphi}_h) \,\mathrm{d}S, \end{aligned} \tag{12}$$

where $\mathcal{B}(\cdot): \mathbb{R}^{d+2} \to \mathbb{R}^{d+2}$ is the operator of the boundary condition given by

$$\mathcal{B}(\boldsymbol{w}_h) := (\rho|_\Gamma, 0, \dots, 0, \rho|_\Gamma \theta|_\Gamma) \text{ for } \Gamma \in \mathscr{F}_h^W, \tag{13}$$

$$\mathcal{B}(\boldsymbol{w}_h) := LRP(\boldsymbol{w}_h|_\Gamma^{(+)}, \boldsymbol{w}_D, \boldsymbol{n}_\Gamma) \text{ for } \Gamma \in \mathscr{F}_h^{io}, \tag{14}$$

where LRP is given by (10). Moreover, the operator $\mathcal{V}: \mathbb{R}^{d+2} \to \mathbb{R}^{d+2}$ is given by $\mathcal{V}(\boldsymbol{\varphi}) = (0, \varphi_2, \dots, \varphi_{d+1}, 0)$ for $\boldsymbol{\varphi} = (\varphi_1, \varphi_2, \dots, \varphi_{d+1}, \varphi_{d+2})$. The role of $\mathcal{V}$ is to penalize only the components of $\boldsymbol{w}$, for which the Dirichlet boundary conditions are prescribed on fixed walls. Moreover, the penalty parameter σ is chosen by

$$\sigma|_\Gamma = C_W / (\mathrm{diam}(\Gamma)\,\mathrm{Re}), \quad \Gamma \in \mathscr{F}_h, \tag{15}$$

where Re is the Reynolds number of the flow, and $C_W > 0$ is a suitable constant which guarantees the convergence of the method.

3.3.4 Fully implicit BDF-DGFE discretization

In order to simplify the notation, for $\boldsymbol{w}_h, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p$, we put

$$\boldsymbol{c}_h\left(\boldsymbol{w}_h, \boldsymbol{\varphi}_h\right) := \boldsymbol{a}_h\left(\boldsymbol{w}_h, \boldsymbol{\varphi}_h\right) + \boldsymbol{b}_h\left(\boldsymbol{w}_h, \boldsymbol{\varphi}_h\right) + \boldsymbol{J}_h^\sigma\left(\boldsymbol{w}_h, \boldsymbol{\varphi}_h\right), \tag{16}$$

The form $\boldsymbol{c}_h$ makes sense also for functions from $H^2(\Omega, \mathscr{T}_h) := \{\boldsymbol{\varphi}; \boldsymbol{\varphi}|_K \in (H^2(K))^{d+2}\ \forall\, K \in \mathscr{T}_h\}$ where $H^2(K)$ is the standard Sobolev space over K.

It is possible to show (see, e.g., [10, 11]) that if $\boldsymbol{w} : \Omega \times (0,T) \to \mathbb{R}^{d+2}$ is a sufficiently regular function satisfying the Navier-Stokes equations (1) and the corresponding initial and boundary conditions, then

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{w}, \boldsymbol{\varphi}\right)_{0,\Omega} + \boldsymbol{c}_h\left(\boldsymbol{w}, \boldsymbol{\varphi}\right) = 0 \quad \forall \boldsymbol{\varphi} \in \boldsymbol{S}_h^p, \tag{17}$$

where $(\cdot,\cdot)_{0,\Omega}$ denotes the L^2-scalar product over Ω.

Now, we introduce the space semi-discretization of (CFP). Let $C^1([0,T]; \boldsymbol{S}_h^p)$ denote the space of continuously differentiable mappings of the interval $[0,T]$ into $\boldsymbol{S}_h^p$.

Definition 3.1. *A function $\boldsymbol{w}_h \in C^1([0,T]; \boldsymbol{S}_h^p)$ is called the* space semi-discrete solution *of (CFP), if*

$$\left(\frac{\partial \boldsymbol{w}_h(t)}{\partial t}, \boldsymbol{\varphi}_h\right)_{0,\Omega} + \boldsymbol{c}_h(\boldsymbol{w}_h(t), \boldsymbol{\varphi}_h) = 0 \qquad \forall\, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p\ \forall\, t \in (0,T), \tag{18a}$$

$$\&\quad \boldsymbol{w}_h(0) = \boldsymbol{w}_{h,0}, \tag{18b}$$

where $\boldsymbol{w}_{h,0} \in \boldsymbol{S}_h^p$ denotes the $\boldsymbol{S}_h^p$-approximation of the initial condition.

Problem (18) represents a system of ordinary differential equations (ODEs) for $\boldsymbol{w}_h(t)$, which has to be discretized in time by a suitable method. Since these ODEs represent a stiff system, it is advantageous to use an implicit time discretization. In order to obtain a sufficiently stable and accurate approximation with respect to the time coordinate, we use the *backward difference formula* (BDF), see, e.g., [21], for the solution the ODE problem (18).

Let $\mathcal{I}_\tau = \{(t_{k-1}, t_k)\}_{k=1}^r$ be a partition of the time interval $(0,T)$ with $0 = t_0 < t_1 < t_2 < \ldots t_r = T$. We put $I_k := (t_{k-1}, t_k)$ and $\tau_k := t_k - t_{k-1},\ k = 1, \ldots r$. Formally, we define $I_0 := (-t_1, t_0)$, this formalism will simplify the later relations. Let $\boldsymbol{w}_h^k \in \boldsymbol{S}_h^p$ denote a piecewise polynomial approximation of $\boldsymbol{w}_h(t_k),\ k = 0, 1, \ldots, r$. We define the following scheme.

Definition 3.2. *The* approximate solution *of (CFP) by the* implicit BDF-DGFE method *is defined as functions $\boldsymbol{w}_h^k \in \boldsymbol{S}_h^p,\ k = 0, \ldots, r$, satisfying the conditions*

$$\frac{1}{\tau_k}\left(\sum_{l=0}^{n} \alpha_{n,l} \boldsymbol{w}_h^{k-l}, \boldsymbol{\varphi}_h\right)_{0,\Omega} + \boldsymbol{c}_h\left(\boldsymbol{w}_h^k, \boldsymbol{\varphi}_h\right) = 0 \quad \forall\, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p,\ k = n, \ldots, r, \tag{19a}$$

$$\boldsymbol{w}_h^0 \ \textit{ is the } L^2\textit{-projection of } \boldsymbol{w}^0 \textit{ in } \boldsymbol{S}_h^p, \tag{19b}$$

$$\boldsymbol{w}_h^l \in \boldsymbol{S}_h^p,\ l = 1, \ldots, n-1 \textit{ are given by a suitable ``less-step'' method}, \tag{19c}$$

where $n \geq 1$ is the degree of the BDF scheme, the BDF coefficients $\alpha_{n,l},\ l = 0, \ldots, n$ depend on time steps $\tau_{k-l},\ l = 0, \ldots, n$.

The n-step BDF-DGFE method (shortly n-BDF-DGFEM) has formally the order of convergence $O(h^p + \tau^n)$ in the $L^2(0,T; H^1(\Omega))$-norm. For $n = 1$ and $n = 2$ these methods are unconditionally stable, and for increasing n they lose more and more stability, for $n > 7$ these methods

	constant time step			variable time step		
	$n=1$	$n=2$	$n=3$	$n=1$	$n=2$	$n=3$
$\alpha_{n,0}$	1	$\frac{3}{2}$	$\frac{11}{6}$	1	$\frac{2\,\theta_k+1}{\theta_k+1}$	$\frac{\theta_k\theta_{k-1}}{\theta_k\theta_{k-1}+\theta_{k-1}+1}+\frac{2\theta_k+1}{\theta_k+1}$
$\alpha_{n,1}$	-1	-2	-3	-1	$-(\theta_k+1)$	$-\frac{(\theta_k+1)(\theta_k\theta_{k-1}+\theta_{k-1}+1)}{\theta_{k-1}+1}$
$\alpha_{n,2}$	—	$\frac{1}{2}$	$\frac{3}{2}$	—	$\frac{\theta_k^2}{\theta_k+1}$	$\frac{\theta_k^2(\theta_k\theta_{k-1}+\theta_{k-1}+1)}{\theta_k+1}$
$\alpha_{n,3}$	—	—	$-\frac{1}{3}$	—	—	$-\frac{(\theta_k+1)\theta_k^2\theta_{k-1}^3}{(\theta_{k-1}+1)(\theta_k\theta_{k-1}+\theta_{k-1}+1)}$

Table 1: Values of $\alpha_{n,l},\ l=0,\dots,n$ for $n=2,3$, $\theta_k := \tau_k/\tau_{k-1},\ k=1,2,\dots,r$

are definitely unstable, see [21, Section III.5]. In practice, we employ the BDF-DGFE scheme for $n=1,2,3$, the values of corresponding coefficients $\alpha_{n,l},\ l=0,\dots,n$ are given in Table 1.

Problem (19) represents a nonlinear algebraic system for each $k=1,\dots,r$ which should be solved by a suitable solver, which is discussed in Section 4. Numerical experiments show that the resulting semi-implicit DGFE method is practically unconditionally stable, i.e., the size of the time step can be chosen very large, see [11].

Remark 3.3. *In practice, we realise relation* (19c) *in such a way, that for* $l=1,\dots,n-1$ *we employ the* l*-step BDF-DGFE scheme, i.e., the one step BDF at first time step, the two steps BDF at second time step, etc. For simplicity, we formally replace* (19a) *and* (19c) *by*

$$\frac{1}{\tau_k}\left(\sum_{l=0}^{n}\alpha_{n,l}\boldsymbol{w}_h^{k-l},\boldsymbol{\varphi}_h\right)_{0,\Omega}+\boldsymbol{c}_h\left(\boldsymbol{w}_h^k,\boldsymbol{\varphi}_h\right)=0\quad\forall\boldsymbol{\varphi}_h\in\boldsymbol{S}_h^p,\ k=1,\dots,r. \tag{20}$$

4 Solution strategy

In this section, we deal with an efficient solution strategy of the nonlinear discrete problem (19).

4.1 Algebraic representation

Let N_h denote the dimension of the piecewise polynomial space $\boldsymbol{S}_h^p$ and

$$B_h := \{\boldsymbol{\varphi}_i(x),\ i=1,\dots,N_h\} \tag{21}$$

denote a set of linearly independent functions forming a basis of $\boldsymbol{S}_h^p$. It is possible to construct a basis B_h as a composition of local bases constructed separately for each $K\in\mathscr{T}_h$. See [14], where one possibility is described in details.

Let $\boldsymbol{w}_h\in\boldsymbol{S}_h^p$ be a piecewise polynomial function. It can be expressed as

$$\boldsymbol{w}_h^k(x)=\sum_{j=1}^{N_h}\xi^{k,j}\boldsymbol{\varphi}_j(x),\quad \boldsymbol{\xi}_k := \{\xi^{k,j}\}_{j=1}^{N_h}\in\mathbb{R}^{N_h},\ k=1,\dots,r, \tag{22}$$

where $\xi^{k,j}\in\mathbb{R},\ j=1,\dots,N_h,\ k=1,\dots,r$ are the basis coefficients. Obviously, (22) defines an isomorphism between $\boldsymbol{w}_h^k\in\boldsymbol{S}_h^p$ and $\boldsymbol{\xi}_k\in\mathbb{R}^{N_h}$.

In order to rewrite the discrete problem (19), we define the vector-valued function $\boldsymbol{F}_h:[\mathbb{R}^{N_h}]^n\times\mathbb{R}^{N_h}\to\mathbb{R}^{N_h}$ by

$$\boldsymbol{F}_h\left(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n;\boldsymbol{\xi}_k\right) := \left\{\frac{1}{\tau_k}\left(\sum_{l=0}^{n}\alpha_{n,l}\boldsymbol{w}_h^{k+1-l},\boldsymbol{\varphi}_i\right)+\boldsymbol{c}_h(\boldsymbol{w}_h^k,\boldsymbol{\varphi}_i)\right\}_{i=1}^{N_h}, \tag{23}$$

where $\boldsymbol{\xi}_{k-l} \in \mathbb{R}^{N_h}$ is the algebraic representation of $\boldsymbol{w}_{h,k-l} \in \boldsymbol{S}_h^p$ for $l = 1, \dots, n$. Therefore, the algebraic representation of the discrete problem (19) reads: for the initial vectors $\boldsymbol{\xi}_0, \boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_{n-1} \in \mathbb{R}^{N_h}$

$$\text{find } \boldsymbol{\xi}_k \in \mathbb{R}^{N_h} \text{ such that } \boldsymbol{F}_h(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n; \boldsymbol{\xi}_k) = \boldsymbol{0}, \quad k = n, \dots, r. \tag{24}$$

The system (24) is strongly nonlinear and its solution is not easy. Nonlinear algebraic systems are usually solved with the aid of the Newton methods, see, e.g., [8], which (usually) requires the evaluation of the Jacobi matrix $D\boldsymbol{F}_h(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n; \boldsymbol{\xi})/D\boldsymbol{\xi}$ or its approximation. A differentiation of $\boldsymbol{F}_h$ can be done either symbolically (which requires a lot of manual work) or numerically (which is time consuming and can be inexact).

Here we present an approach where the Jacobi matrix in the Newton method is replaced by the so-called *flux matrix* developed in the context of the semi-implicit DGFE method in [11, 13, 14].

4.2 Flux matrix

In virtue of (7), (11), (12), for $\bar{\boldsymbol{w}}_h, \boldsymbol{w}_h, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p$, we define the form $\boldsymbol{c}_h^L : \boldsymbol{S}_h^p \times \boldsymbol{S}_h^p \times \boldsymbol{S}_h^p \to \mathbb{R}$

$$\begin{aligned}
&\boldsymbol{c}_h^L(\bar{\boldsymbol{w}}_h, \boldsymbol{w}_h, \boldsymbol{\varphi}_h) \qquad (25)\\
&:= \sum_{\Gamma \in \mathscr{F}_h} \int_\Gamma \boldsymbol{P}^+ \left(\langle \bar{\boldsymbol{w}}_h \rangle, \boldsymbol{n}\right) \boldsymbol{w}_h|_\Gamma^{(+)} \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S + \sum_{\Gamma \in \mathscr{F}_h \setminus \mathscr{F}_h^{io}} \int_\Gamma \boldsymbol{P}^- \left(\langle \bar{\boldsymbol{w}}_h \rangle, \boldsymbol{n}\right) \boldsymbol{w}_h|_\Gamma^{(-)} \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S\\
&\quad - \sum_{K \in \mathscr{T}_h} \int_K \sum_{s=1}^d \boldsymbol{A}_s(\bar{\boldsymbol{w}}_h) \boldsymbol{w}_h \cdot \frac{\partial \boldsymbol{\varphi}_h}{\partial x_s} \,\mathrm{d}x + \sum_{K \in \mathscr{T}_h} \int_K \sum_{s,k=1}^d \left(\boldsymbol{K}_{s,k}(\bar{\boldsymbol{w}}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k}\right) \cdot \frac{\partial \boldsymbol{\varphi}_h}{\partial x_s} \,\mathrm{d}x\\
&\quad - \sum_{\Gamma \in \mathscr{F}_h^I \cup \mathscr{F}_h^{io}} \int_\Gamma \sum_{s=1}^d \left\langle \sum_{k=1}^d \boldsymbol{K}_{s,k}(\bar{\boldsymbol{w}}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k} \right\rangle n_s \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S\\
&\quad - \sum_{\Gamma \in \mathscr{F}_h^W} \int_\Gamma \sum_{s,k=1}^d \boldsymbol{K}_{s,k}^W(\bar{\boldsymbol{w}}_h) \frac{\partial \boldsymbol{w}_h}{\partial x_k} n_s \cdot \boldsymbol{\varphi}_h \,\mathrm{d}S\\
&\quad + \sum_{\Gamma \in \mathscr{F}_h^I} \int_\Gamma \sigma [\![\boldsymbol{w}_h]\!] \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S + \sum_{\Gamma \in \mathscr{F}_h^{io}} \int_\Gamma \sigma \boldsymbol{w}_h \cdot \boldsymbol{\varphi}_h \,\mathrm{d}S + \sum_{\Gamma \in \mathscr{F}_h^W} \int_\Gamma \sigma \boldsymbol{w}_h \cdot \mathcal{V}(\boldsymbol{\varphi}_h) \,\mathrm{d}S
\end{aligned}$$

and the form $\boldsymbol{d}_h : \mathbb{R}^{N_h} \times \mathbb{R}^{N_h} \to \mathbb{R}$ by

$$\begin{aligned}
\boldsymbol{d}_h(\bar{\boldsymbol{w}}_h, \boldsymbol{\varphi}_h) := &- \sum_{\Gamma \in \mathscr{F}_h^{io}} \int_\Gamma \boldsymbol{P}^- \left(\langle \bar{\boldsymbol{w}}_h \rangle, \boldsymbol{n}\right) \bar{\boldsymbol{w}}_h|_\Gamma^{(-)} \cdot [\![\boldsymbol{\varphi}_h]\!] \,\mathrm{d}S \qquad (26)\\
&+ \sum_{\Gamma \in \mathscr{F}_h^{io}} \int_\Gamma \sigma \mathcal{B}(\bar{\boldsymbol{w}}_h) \cdot \boldsymbol{\varphi}_h \,\mathrm{d}S + \sum_{\Gamma \in \mathscr{F}_h^W} \int_\Gamma \sigma \mathcal{B}(\bar{\boldsymbol{w}}_h) \cdot \mathcal{V}(\boldsymbol{\varphi}_h) \,\mathrm{d}S.
\end{aligned}$$

Obviously, due to relations (7), (11), (12) and (25) – (26), we have the consistency between forms $\boldsymbol{c}_h$ and $\boldsymbol{c}_h^L$, namely

$$\boldsymbol{c}_h(\boldsymbol{w}_h, \boldsymbol{\varphi}_h) = \boldsymbol{c}_h^L(\boldsymbol{w}_h, \boldsymbol{w}_h, \boldsymbol{\varphi}_h) - \boldsymbol{d}_h(\boldsymbol{w}_h, \boldsymbol{\varphi}_h) \qquad \forall \boldsymbol{w}_h, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h^p. \tag{27}$$

Furthermore, form $\boldsymbol{c}_h^L$ is linear with respect to its second (and third) argument.

Using the notation from Section 4.1, we define the *flux matrix*

$$\mathbb{C}_h\left(\bar{\boldsymbol{\xi}}\right) := \left\{ \frac{\alpha_{n,0}}{\tau_k} \left(\boldsymbol{\varphi}_j, \boldsymbol{\varphi}_i\right)_{0,\Omega} + \boldsymbol{c}_h^L(\bar{\boldsymbol{w}}_h, \boldsymbol{\varphi}_j, \boldsymbol{\varphi}_i) \right\}_{i,j=1}^{N_h} \tag{28}$$

and the vector

$$\boldsymbol{q}_h\left(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n, \bar{\boldsymbol{\xi}}\right) := \left\{ -\frac{1}{\tau_k} \left(\sum_{i=1}^n \alpha_{n,i} \boldsymbol{w}_h^{k-l}, \boldsymbol{\varphi}_i \right)_{0,\Omega} + \boldsymbol{d}_h(\bar{\boldsymbol{w}}_h, \boldsymbol{\varphi}_i) \right\}_{i=1}^{N_h}, \tag{29}$$

where $\bar{\boldsymbol{\xi}} \in \mathbb{R}^{N_h}$ and $\boldsymbol{\xi}_{k-l} \in \mathbb{R}^{N_h},\ l = 1, \ldots, n$ are the algebraic representation of $\bar{\boldsymbol{w}}_h \in \boldsymbol{S}_h^p$ and $\boldsymbol{w}_h^{k-l} \in \boldsymbol{S}_h^p,\ l = 1, \ldots, n$, respectively. Finally, using (23) and (28) –(29), we have

$$\boldsymbol{F}_h(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n; \boldsymbol{\xi}_k) = \mathbb{C}_h(\boldsymbol{\xi}_k)\boldsymbol{\xi}_k - \boldsymbol{q}_h(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n, \boldsymbol{\xi}_k). \tag{30}$$

Let us note that the flux matrix $\mathbb{C}_h$ has a block structure and it is sparse. In virtue of (25) we easily find that each block-row of $\mathbb{C}_h$ corresponds to one $K \in \mathscr{T}_h$ and it contains a diagonal block and several off-diagonal blocks. Each off-diagonal block corresponds to one face $\Gamma \in \mathscr{F}_h$. Obviously, the sparsity of $\mathbb{C}_h$ is identical to the sparsity of the Jacobi matrix $D\boldsymbol{F}_h(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n; \boldsymbol{\xi})/D\boldsymbol{\xi}$. Therefore, in the following Newton-like method, we use $\mathbb{C}_h$ as the approximation of $D\boldsymbol{F}_h(\{\boldsymbol{\xi}_{k-l}\}_{l=1}^n; \boldsymbol{\xi})/D\boldsymbol{\xi}$. This approximation follows from relation (30), when we fix the arguments of $\mathbb{C}_h$ and $\boldsymbol{q}_h$ and perform the differentiation with respect to $\boldsymbol{\xi}_k$.

Remark 4.1. *We easily find that for the evaluation of each entry of $\boldsymbol{F}_h$ and/or $\mathbb{C}_h$ we need solution $\boldsymbol{w}_h$ from at most two neighboring elements. This is a favorable property which simplifies the parallelization of the algorithm, see Section 5.*

Remark 4.2. *Let us mention computational costs of the evaluation of $\boldsymbol{F}_h$ and $\mathbb{C}_h$. For simplicity, let us consider the case $d = 2$, $p_K = p\ \forall K \in \mathscr{T}_h$ and $\mathscr{T}_h$ conforming triangular grid. Then $\boldsymbol{F}_h$ has $N_h = \#\mathscr{T}_h(p+1)(p+1)/2$ entries and $\mathbb{C}_h$ has approximately $4\#\mathscr{T}_h((p+1)(p+1)/2)^2$ non-vanishing entries. Therefore, an evaluation of $\boldsymbol{F}_h$ is approximately $2(p+1)(p+2)$ cheaper than an evaluation of $\mathbb{C}_h$.*

4.3 Iterative algorithm

To determine solution $\boldsymbol{\xi}_k$ of the system (24), we employ a damped Newton-like method which generates a sequence of approximations $\boldsymbol{\xi}_k^n,\ n = 0, 1, \ldots$ to the actual numerical solution $\boldsymbol{\xi}_k$ using the following algorithm. Given an iterate $\boldsymbol{\xi}_k^n$, the update $\boldsymbol{d}^n$ of $\boldsymbol{\xi}_k^n$ to get to the next iterate

$$\boldsymbol{\xi}_k^{n+1} := \boldsymbol{\xi}_k^n + \lambda^n \boldsymbol{d}^n \tag{31}$$

is defined by: find $\boldsymbol{d}^n \in \mathbb{R}^{N_h}$ such that

$$\mathbb{C}_h(\boldsymbol{\xi}_k^n)\boldsymbol{d}^n = -\boldsymbol{F}_h(\boldsymbol{\xi}_k^n), \tag{32}$$

where $\mathbb{C}_h$ is the flux matrix given by (30) and $\lambda^n \in (0, 1]$ is a damping parameter which ensures convergence of (31) – (32) in case when the initial guess $\boldsymbol{\xi}_k^0$ is far from the solution of (24).

4.3.1 Choice of the damping parameter

We start from the value $\lambda^n = 1$ and evaluate a monitoring function $\theta^n := \left\|\boldsymbol{F}_h(\boldsymbol{\xi}_k^{n+1})\right\| / \|\boldsymbol{F}_h(\boldsymbol{\xi}_k^n)\|$. If $\theta^n < 1$ we proceed to the next Newton iteration. Otherwise, we put $\lambda^n := \lambda^n/2$ and repeat the actual Newton iteration. Analysis of the convergence of this simplified Newton method and the monitoring function can be found in [8].

4.3.2 Update of the flux matrix

Obviously, it is not necessary to update the flux matrix $\mathbb{C}_h(\boldsymbol{\xi}_k^n)$ at each Newton iteration $n = 1, 2, \ldots$ and each time level $k = 1, \ldots, r$. In virtue of Remark 4.2, it is much cheaper to evaluate $\boldsymbol{F}_h$ than $\mathbb{C}_h$. Therefore, it is more efficient to perform more Newton iterations than to update $\mathbb{C}_h$. In practice, we update $\mathbb{C}_h$ either the damping parameter λ achieves a minimal prescribed value (using the algorithm described in Section 4.3.1) or the prescribed maximal number of Newton iteration was achieved.

4.3.3 Termination of iterative process

The iterative process (31) – (32) is terminated if a suitable *algebraic stopping criterion* is achieved, e.g.,

$$\|\boldsymbol{F}_h(\boldsymbol{\xi}_k^n)\| \leq \mathrm{TOL}, \tag{33}$$

where $\|\cdot\|$ and TOL are a given norm and a given tolerance, respectively.

5 Parallelization

5.1 Implementation issues

DGFE method can be easily parallelized since the corresponding stencils do not grow in size with increasing order. In the following, we briefly describe the parallel implementation of method (19), a) – c) which is based on the decomposition of the mesh to the particular processors of a cluster with a distributed memory.

First, we decompose triganulation $\mathscr{T}_h$ into several subtriagulations $\mathscr{T}_h^i$, $i = 1, \ldots, \pi$, where π is the number of the processors. Let N_K denote the local number of degree of freedom per element, i.e, $N_K = \frac{(d+2)}{d!}(\Pi_{j=1}^d (p_K + j))$, $K \in \mathscr{T}_h$, where $'_K$ denotes the degree of polynomial approximation over $K \in \mathscr{T}_h$. In order to equilibrate the computational work and the use of the memory for each processor, we requires that

$$\sum_{K \in \mathscr{T}_h^i} (N_K)^2 \approx \frac{1}{\pi} \sum_{K \in \mathscr{T}_h} (N_K)^2, \qquad i = 1, \ldots, \pi, \tag{34}$$

since the count of computational operations and the amount of the memory corresponding to matrix $\mathbb{C}$ is proportional to the square of the degree of freedom. If the fixed degree of polynomial approximation is used for all $K \in \mathscr{T}_h$ then relation (34) reduces to $\#\mathscr{T}_h^i \approx \#\mathscr{T}_h$, $i = 1, \ldots, \pi$, where $\#\mathscr{T}_h$ denotes the number of elements of $\mathscr{T}_h$. Moreover, in order to minimize the communication between processor we minimize the amount of $\Gamma \in \mathscr{F}_h$ such that $\Gamma \subset \partial K \cap \partial K'$, $K \in \mathscr{T}_h^i$, $K' \in \mathscr{T}_h^j$ and $i \neq j$, see Remark 4.1. For the mesh decomposition we employ software package METIS [26], see also [25].

Since the memory is distributed, we accociate to each element $i = 1, \ldots, \pi$ the rows of $\boldsymbol{F}_h$ and $\mathbb{C}_h$ corresponding to $K \in \mathscr{T}_h^i$, $i = 1, \ldots, \pi$. Figure 1 illustrates a decomposition of a fictitious mesh $\mathscr{T}_h$ and the corresponding matrix $\mathbb{C}$ into three processors.

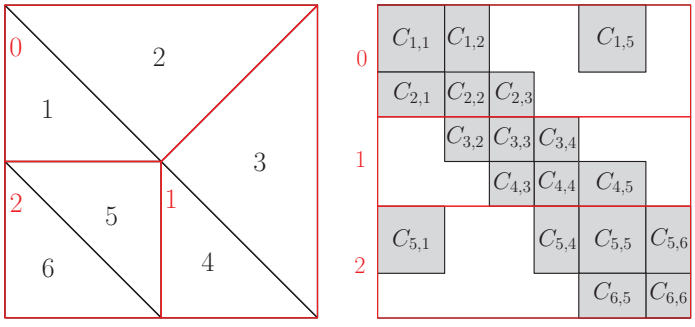

Figure 1: Decomposition of triangulation $\mathscr{T}_h$ having 6 elements (left) and the corresponding matrix $\mathbb{C}$ (right) into three parts (processors) indexed 0,1,2.

For the evaluation of off-diagonal blocks of $\mathbb{C}$ the information from two neighboring elements $K \in \mathscr{T}_h$ corresponding two different processors is also requires. In order to save the computational time, we employ the ability of a simultaneous computation and processor communication.

Therefore, each processor $i = 1, \ldots, \pi$ sends and receives data during evaluation of the volume integrals and face integrals lying in interior of $\mathscr{T}_h^i$.

Finally, the corresponding linear algebraic systems are solved with the aid of PETSc (Portable, Extensible Toolkit for Scientific Computation, [1]) library. We tested GMRES and BiCGStab solved together with the block Jacobi and the additive Schwarz preconditioners which are implemented in PETSc.

5.2 Numerical experiments

In this section we present the computational performance of the parallel implementation of (19) described above. The presented results correspond to the subsonic viscous flow around NACA0012 profiles with $M = 0.5$, $\alpha = 0°$, $Re = 5000$. We employ P_l, $l = 1, 2, 3$ polynomial approximation on meshes $\mathscr{T}_{h1}$ and $\mathscr{T}_{h2}$ having 6 876 and 29 040 elements. The corresponding number of degree of freedom is given in Table 2.

	$\mathscr{T}_{h1}$	$\mathscr{T}_{h2}$
P_1	82 512	348 480
P_2	165 024	696 960
P_3	275 040	1 161 600

Table 2: Number of degree of freedom for each computation

In the following we study ....

5.2.1 Mesh decomposition

We already mention that triangulation $\mathscr{T}_h$ is decomposed on sub-triangulations $\mathscr{T}_h^i$, $i = 1, \ldots, \pi$ with the aid of METIS software. The quality of decomposition is characterized by the *load balancing* and the relative number of *ghost cells*. We characterize the load balancing by the minimal and maximal number of elements of meshes $\mathscr{T}_h^i$ for $i = 1, \ldots, \pi$. Moreover, the relative number of ghost cells is the ratio of GC (=the number of the elements having an edge on the boundary between $\mathscr{T}_h^i$ and $\mathscr{T}_h^j$, $i \neq j$) and the number of processors.

Table 3 shows the results of the mesh decomposition of grids $\mathscr{T}_{h1}$ and $\mathscr{T}_{h2}$, namely the values $\min_{i=1,\ldots,\pi} \#\mathscr{T}_h^i$, $\max_{i=1,\ldots,\pi} \#\mathscr{T}_h^i$, number of ghost cells GC and the relative number of ghost cells GC/π, for the number of processors $\pi \in \{1,\ 2,\ 4,\ 16,\ 32\}$. We easily observe very good load balancing and the decreasing value of the relative number of ghost cells for increasing number of processors. Finally, Figure2 illustrates the mesh decomposition of $\mathscr{T}_{h2}$ between 16 and 32 processors.

	$\mathscr{T}_{h1}$				$\mathscr{T}_{h2}$			
π	$\min \#\mathscr{T}_h^i$	$\max \#\mathscr{T}_h^i$	GC	GC/π	$\min \#\mathscr{T}_h^i$	$\max \#\mathscr{T}_h^i$	GC	GC/π
1	6876	6876	0	0.00 %	29040	29040	0	0.00 %
2	3438	3438	45	0.76 %	14520	14520	103	0.36 %
4	1719	1719	107	0.78 %	7260	7260	214	0.42 %
8	859	860	195	0.71 %	3630	3630	415	0.36 %
16	428	431	311	0.57 %	1814	1816	669	0.29 %
32	214	216	518	0.47 %	907	908	1101	0.24 %

Table 3: Quality of mesh decomposition, *load balancing* characterized by $\min_{i=1,\ldots,\pi} \#\mathscr{T}_h^i$ and $\max_{i=1,\ldots,\pi} \#\mathscr{T}_h^i$, number of *ghost cells* GC and the relative number of GC per processor GC/π

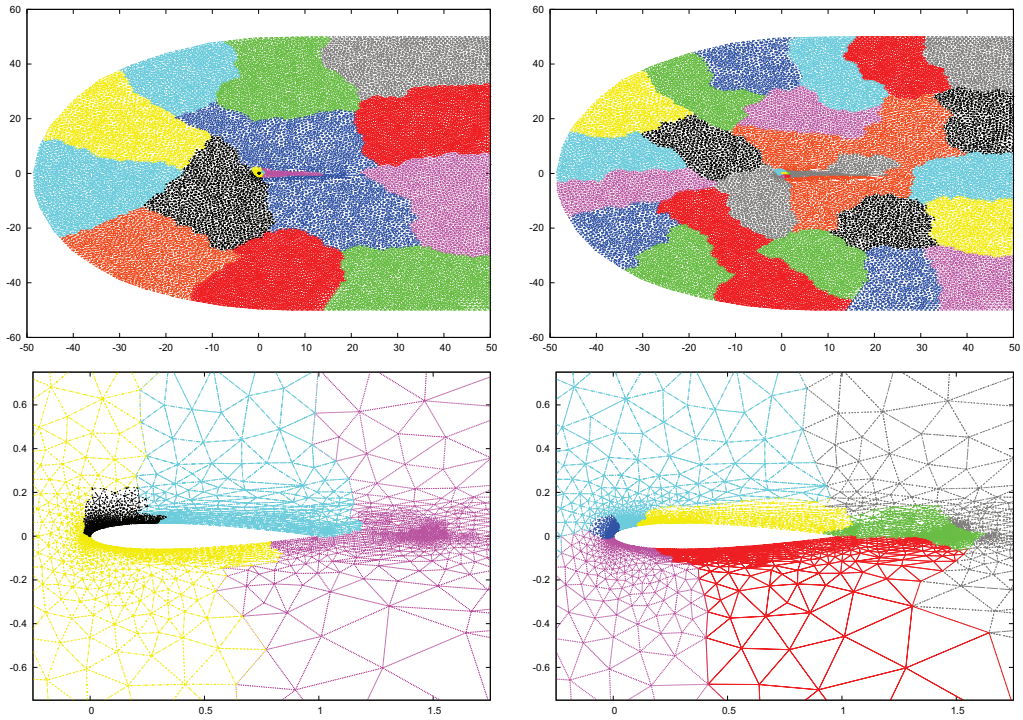

Figure 2: Mesh decomposition of $\mathscr{T}_{h2}$ between 16 (left) and 32 (right) processors, the global view (top) and details around profile (bottom)

	GMRES	BiCGStab
block Jacobi (BJ)	160 MB	150 MB
additive Schwarz (AS)	201 MB	195 MB

Table 4: Comparison of memory requirement for one processor

5.2.2 Linear solvers and preconditioners

We carried out a comparison of the computational performance of linear iterative solvers with preconditioner implemented in PETSc. Particularly, we employ GMRES and BiCGStab solvers in combination with block Jacobi (BJ) and additive Schwarz (AS) preconditioners. We presented experiments achieved by P_2 approximation on mesh $\mathscr{T}_{h2}$ with the aid of 16 processors. Figure 3 show the convergence to the steady-state solution for all four combinations of solvers and preconditioners with respect to the computational time and the number of time steps. In AS preconditioner, we choose the overlap $\delta = 1$. We simply observe that the difference between solvers and preconditioners is not essential with respect to the computational time as well as the number of time steps. Slightly better is the BiCGStab solver with AS preconditioner.

Moreover, we investigate the influence of the size of the overlapping in AS preconditioner (δ) to the steady-state convergence. Figure 4 shows the convergence to the steady state for the values $\delta = 0$, $\delta = 1$, $\delta = 3$ and $\delta = 5$. We observe that the fastest is the smaller overlapping with $\delta = 0$ and $\delta = 1$ since the lager values of δ requires more time for processors communications.

Finally, Table 4 shows the maximal memory requirement for one processor. Obviously, BJ preconditioner need less memory. Since the computational performance of all tested method is similar, in the following we employ the combination with the smallest memory requirements, i.e., BiCGStab with block Jacobi preconditioner.

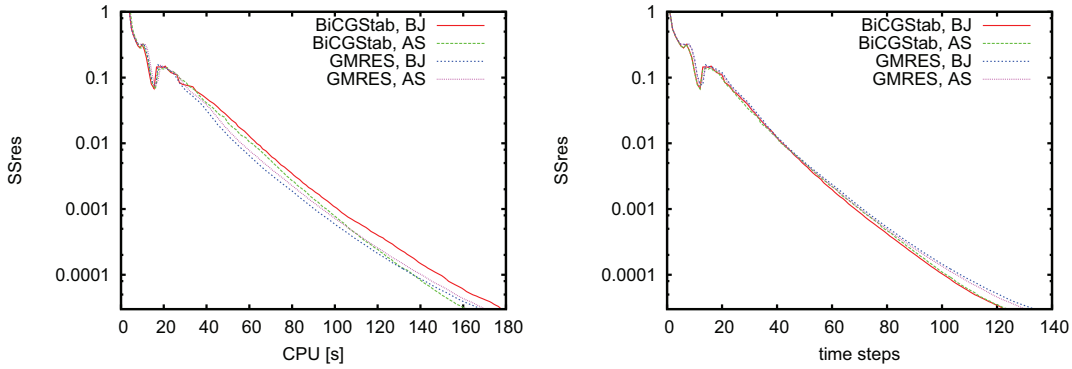

Figure 3: Convergence to the steady-state solution for all four combinations of solvers and preconditioners with respect to the computational time (left) and the number of time steps (right)

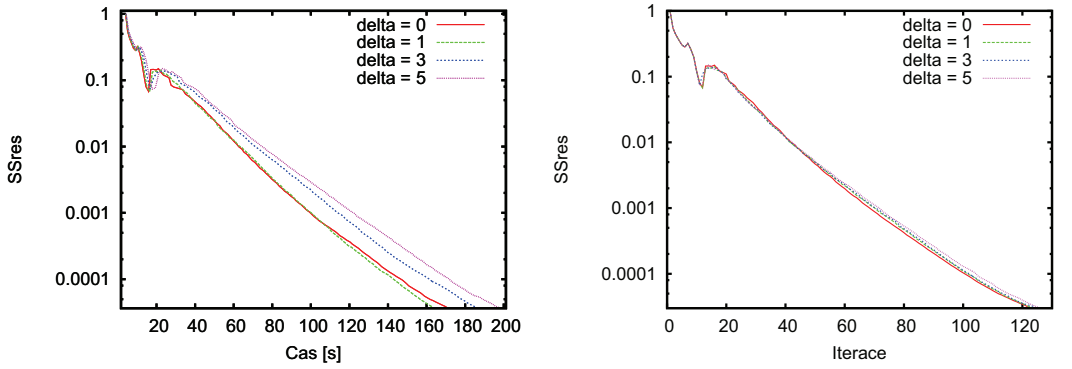

Figure 4: Dependence of the overlapping in AS preconditioner to the convergence to steady-state solution with respect to the computational time (left) and the number of time steps (right)

5.2.3 Efficiency of the parallelization

We study the computational time T_π necessary to achieve the steady state solution in the dependence on the number of processors π. We define the acceleration $S_\pi = T_1/T_\pi$ and the efficiency $E_\pi = S_\pi \pi$. In ideal case, $S_\pi = \pi$ and hence $E_\pi = 1$.

Table 5 shows the computational times T_π (in seconds), acceleration S_π and efficiency E_π for $\pi \in \{1,\ 2,\ 4,\ 16,\ 32\}$ achieved with the aid of $P_k,\ k = 1, 2, 3$ approximation on grids $\mathscr{T}_{h1}$ and $\mathscr{T}_{h2}$.

Moreover, Figure 5 shows the acceleration S_π and Figure 6 the efficiency E_π.

We observe that for the small number of processors $(1 - -4)$, the efficiency is high even for coarser grids. For increasing number of processors, the efficiency is decreasing for the increasing number of processors in the dependence on number of elements of $\mathscr{T}_h$. It is caused by some non-negligible computational time necessary for the pre- and post-processing independent of π. Obviously, for the finner $\mathscr{T}_h$ and higher degree of polynomial approximation, the decrease of efficiency is slower since the pre- and post-processing are less substantial.

Moreover, the decrease of efficiency is less essential for higher degree of polynomial approximation since the computational time of the pre- and post-processing depends mostly on the number of elements of $\mathscr{T}_h$ and not so strongly on the degree of polynomial approximation. In order to eliminate the decrease of efficiency for increasing π, we should parallelize also the pre- and post-processing

Furthermore, let us note that it some situations $E_\pi > 1$. It can be caused from two reasons. Firstly, we should take into account the numbering of matrix block of $\mathbb{C}_h$ is different for different π.

	$\mathscr{T}_{h1}$, P_1			$\mathscr{T}_{h2}$, P_1		
π	T_π	S_π	E_π	T_π	S_π	E_π
1	102.43 s	1.00	1.00	539.73 s	1.00	1.00
2	50.75 s	2.02	1.01	258.16 s	2.09	1.05
4	25.90 s	3.95	0.99	131.49 s	4.10	1.03
8	15.07 s	6.80	0.85	65.80 s	8.20	1.03
16	9.79 s	10.46	0.65	35.21 s	15.33	0.96
32	7.77 s	13.19	0.41	26.75 s	20.18	0.63
	$\mathscr{T}_{h1}$, P_2			$\mathscr{T}_{h2}$, P_2		
π	T_π	S_π	E_π	T_π	S_π	E_π
1	357.20 s	1.00	1.00	1846.29 s	1.00	1.00
2	179.08 s	1.99	1.00	886.30 s	2.08	1.04
4	90.19 s	3.96	0.99	461.60 s	4.00	1.00
8	49.13 s	7.27	0.91	238.03 s	7.76	0.97
16	27.01 s	13.22	0.83	131.04 s	14.09	0.88
32	16.33 s	21.87	0.68	79.88 s	23.11	0.72
	$\mathscr{T}_{h1}$, P_3			$\mathscr{T}_{h2}$, P_3		
π	T_π	S_π	E_π	T_π	S_π	E_π
1	1278.27 s	1.00	1.00	5988.53 s	1.00	1.00
2	651.42 s	1.96	0.98	3026.76 s	1.98	0.99
4	310.50 s	4.12	1.03	1526.50 s	3.92	0.98
8	156.58 s	8.16	1.02	760.79 s	7.87	0.98
16	87.28 s	14.65	0.92	395.66 s	15.13	0.94
32	50.07 s	25.53	0.80	206.64 s	28.98	0.91

Table 5: Computational times T_π, acceleration S_π and efficiency E_π for $\pi \in \{1,\ 2,\ 4,\ 16,\ 32\}$ achieved with the aid of $P_k,\ k = 1, 2, 3$ approximation on grids $\mathscr{T}_{h1}$ and $\mathscr{T}_{h2}$

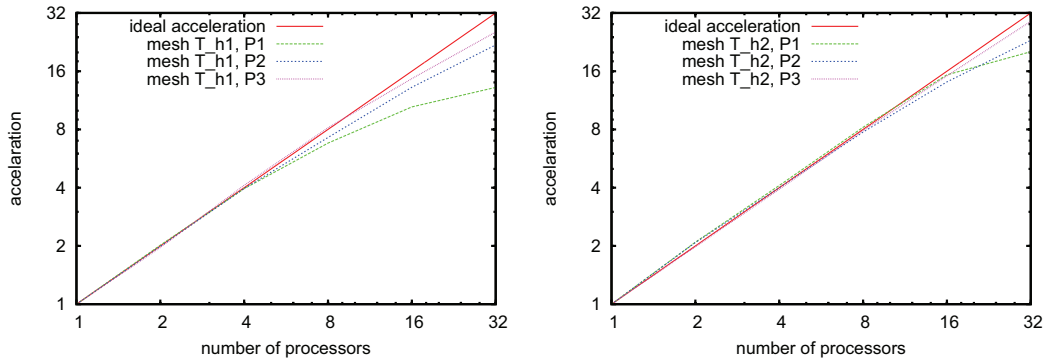

Figure 5: Acceleration S_π on meshes $\mathscr{T}_{h1}$ (left) and $\mathscr{T}_{h2}$ (right)

Hence, although we solve the same linear system, the iterative solvers achieve generally different numerical solution for different π, i.e., the iterative process can be stopped earlier. Secondly, we should take into the *cache effect* when more the faster cache memory is employed for higher number of processors.

Finally, let us mention the memory requirement of our parallel implementation The highest amount of memory is requires for the linear iterative solvers from the PETSc library. Let $m_\pi^i,\ i = 1, \ldots, \pi$ denotes the number of allocated memory on th i^{th} processor when π processors was

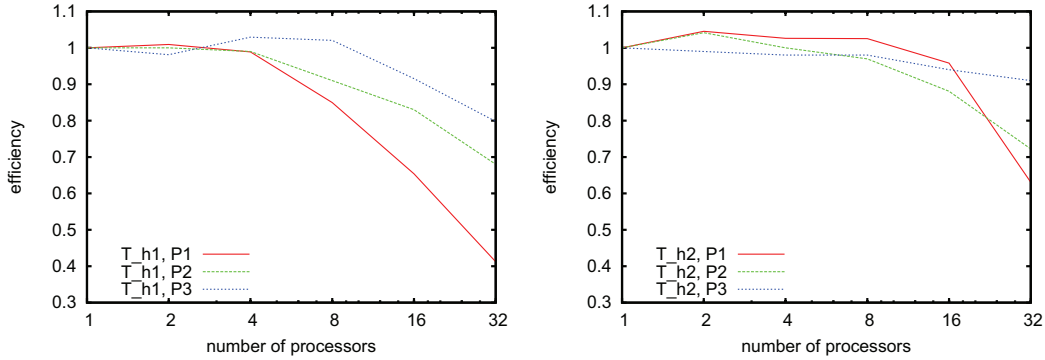

Figure 6: Efficiency E_π on meshes $\mathscr{T}_{h1}$ (left) and $\mathscr{T}_{h2}$ (right)

employed. Table 6 shows the quantities

$$\overline{m}_\pi := \max_{i=1,\dots,\pi} m_\pi^i, \quad \underline{m}_\pi := \min_{i=1,\dots,\pi} m_\pi^i, \quad m_\pi^{\mathrm{tot}} := \sum_{i=1}^{\pi} m_\pi^i, \quad M_\pi := \frac{\overline{m}_1}{\overline{m}_\pi} \tag{35}$$

The *scaling factor* measures the decrease of the memory requirement per one processor for increasing π. In ideal case, $M_\pi = \pi$. Table 6 shows the values defined in (35) for $P_k,\ k = 1, 2, 3$ approximation on grids $\mathscr{T}_{h1}$ and $\mathscr{T}_{h2}$.

	$\mathscr{T}_{h1}$, P_1					$\mathscr{T}_{h2}$, P_1				
p	$\overline{m}_\pi$	$\underline{m}_\pi$	$\frac{\overline{m}_\pi}{\underline{m}_\pi}$	m_π^{tot}	M_π	$\overline{m}_\pi$	$\underline{m}_\pi$	$\frac{\overline{m}_\pi}{\underline{m}_\pi}$	m_π^{tot}	M_π
1	111	111	1.00	111	1.00	425	425	1.00	425	1.00
2	71	71	1.00	146	1.57	249	249	1.00	497	1.71
4	50	49	1.04	197	2.20	159	156	1.02	629	2.67
8	39	37	1.05	299	2.85	115	111	1.04	892	3.70
16	33	32	1.04	521	3.35	92	89	1.03	1441	4.63
32	31	30	1.05	973	3.58	87	77	1.13	2568	4.88
	$\mathscr{T}_{h1}$, P_2					$\mathscr{T}_{h2}$, P_2				
p	$\overline{m}_\pi$	$\underline{m}_\pi$	$\frac{\overline{m}_\pi}{\underline{m}_\pi}$	m_π^{tot}	M_π	$\overline{m}_\pi$	$\underline{m}_\pi$	$\frac{\overline{m}_\pi}{\underline{m}_\pi}$	m_π^{tot}	M_π
1	336	336	1.00	336	1.00	1260	1260	1.00	1260	1.00
2	183	183	1.00	365	1.85	669	668	1.00	1337	1.88
4	105	104	1.01	418	3.20	399	396	1.01	1589	3.15
8	67	65	1.03	525	5.00	234	231	1.02	1851	5.37
16	47	46	1.03	736	7.15	150	148	1.02	2374	8.38
32	38	37	1.02	1189	8.91	117	109	1.07	3512	10.80
	$\mathscr{T}_{h1}$, P_3					$\mathscr{T}_{h2}$, P_3				
p	$\overline{m}_\pi$	$\underline{m}_\pi$	$\frac{\overline{m}_\pi}{\underline{m}_\pi}$	m_π^{tot}	M_π	$\overline{m}_\pi$	$\underline{m}_\pi$	$\frac{\overline{m}_\pi}{\underline{m}_\pi}$	m_π^{tot}	M_π
1	779	779	1.00	779	1.00	3259	3259	1.00	3259	1.00
2	404	403	1.00	808	1.93	1659	1657	1.00	3316	1.96
4	237	235	1.01	944	3.29	865	861	1.00	3452	3.77
8	133	131	1.01	1050	5.87	470	465	1.01	3736	6.94
16	79	78	1.02	1258	9.81	289	286	1.01	4591	11.28
32	54	52	1.02	1690	14.55	180	178	1.01	5730	18.07

Table 6: The memory requirement (in MB) for $P_k,\ k = 1, 2, 3$ approximation on grids $\mathscr{T}_{h1}$ and $\mathscr{T}_{h2}$, coefficient defined in (35) .

We observe that the memory is well distributed among the processors since $\overline{m}_\pi/\underline{m}_\pi$ is close

to one. On the other hand, the parallel implementation is far from an ideal one since the total amount of memory m_π^{tot} is increasing for increasing π. Although the scaling factor M_π is far from the ideal value ($=\pi$), it is closer to the ideal value for the increasing number of elements of $\mathscr{T}_h$ and for the increasing degree of polynomial approximation.

6 Conclusion

We deal with the semi-implicit discontinuous Galerkin method for the numerical solution of viscous compressible flows, which leads to the solution of a sequence of linear algebraic systems. We developed an efficient solution strategy for steady-state flow regimes. The presented numerical experiments show that the parallelization gives almost linear acceleration of the computation with respect to the number of processors.

References

[1] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. *PETSc – Portable, Extensible Toolkit for Scientific Computation*, 2011. `<http://www.mcs.anl.gov/petsc/>`.

[2] F. Bassi and S. Rebay. A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations. *J. Comput. Phys.*, 131:267–279, 1997.

[3] F. Bassi and S. Rebay. A high order discontinuous Galerkin method for compressible turbulent flow. In B. Cockburn, G. E. Karniadakis, and C.-W. Shu, editors, *Discontinuous Galerkin Method: Theory, Computations and Applications*, Lecture Notes in Computational Science and Engineering 11, pages 113–123. Springer-Verlag, 2000.

[4] C. E. Baumann and J. T. Oden. A discontinuous *hp* finite element method for the Euler and Navier-Stokes equations. *Int. J. Numer. Methods Fluids*, 31(1):79–95, 1999.

[5] P. G. Ciarlet. *The Finite Elements Method for Elliptic Problems*. North-Holland, Amsterdam, New York, Oxford, 1979.

[6] B. Cockburn, S. Hou, and C. W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element for conservation laws IV: The multi-dimensional case. *Math. Comp.*, 54:545–581, 1990.

[7] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[8] P. Deuflhard. *Newton Methods for Nonlinear Problems*, volume 35 of *Springer Series in Computational Mathematics*. Springer, 2004.

[9] L. T. Diosady and D. L. Darmofal. Preconditioning methods for discontinuous Galerkin solutions of the Navier-Stokes equations. *J. Comput. Phys.*, 228:3917–3935, 2009.

[10] V. Dolejší. On the discontinuous Galerkin method for the numerical solution of the Navier–Stokes equations. *Int. J. Numer. Methods Fluids*, 45:1083–1106, 2004.

[11] V. Dolejší. Semi-implicit interior penalty discontinuous Galerkin methods for viscous compressible flows. *Commun. Comput. Phys.*, 4(2):231–274, 2008.

[12] V. Dolejší. Discontinuous Galerkin method for the numerical simulation of unsteady compressible flow. *WSEAS Transactions on Systems*, 5(5):1083–1090, 2006.

[13] V. Dolejší and M. Feistauer. Semi-implicit discontinuous Galerkin finite element method for the numerical solution of inviscid compressible flow. *J. Comput. Phys.*, 198(2):727–746, 2004.

[14] V. Dolejší, M. Holík, and J. Hozman. Efficient solution strategy for the semi-implicit discontinuous Galerkin discretization of the Navier-Stokes equations. *J. Comput. Phys.*, 230:41764200, 2011.

[15] M. Dumbser. Arbitrary high order PNPM schemes on unstructured meshes for the compressible Navier-Stokes equations. *Comput. Fluids*, 39(1):60–76, 2010.

[16] M. Dumbser and C.-D. Munz. Building blocks for arbitrary high-order discontinuous Galerkin methods. *J. Sci. Comput.*, 27:215–230, 2006.

[17] M. Feistauer, J. Felcman, and I. Straškraba. *Mathematical and Computational Methods for Compressible Flow*. Oxford University Press, Oxford, 2003.

[18] M. Feistauer and V. Kučera. On a robust discontinuous Galerkin technique for the solution of compressible flow. *J. Comput. Phys.*, 224(1):208–221, 2007.

[19] M. Feistauer, V. Kučera, and J. Prokopová. Discontinuous Galerkin solution of compressible flow in time dependent domains. *Mathematics and Computers in Simulations*, 80(8):1612–1623, 2010.

[20] G. Gassner, F. Lörcher, and C.-D. Munz. A discontinuous Galerkin scheme based on a spacetime expansion. I. Inviscid compressible flow in one space dimension. *J. Sci. Comput.*, 32(2):175–199, 2007.

[21] E. Hairer, S. P. Norsett, and G. Wanner. *Solving ordinary differential equations I, Nonstiff problems*. Number 8 in Springer Series in Computational Mathematics. Springer Verlag, 2000.

[22] R. Hartmann and P. Houston. Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations. *J. Comput. Phys.*, 183(2):508–532, 2002.

[23] R. Hartmann and P. Houston. Symmetric interior penalty DG methods for the compressible Navier-Stokes equations I: Method formulation. *Int. J. Numer. Anal. Model.*, 1:1–20, 2006.

[24] R. Hartmann and P. Houston. Symmetric interior penalty DG methods for the compressible Navier-Stokes equations II: Goal-oriented a posteriori error estimation. *Int. J. Numer. Anal. Model.*, 3:141–162, 2006.

[25] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998.

[26] G. Karypis and V. Kumar. *METIS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, 2011. `<http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>`.

[27] C. M. Klaij, J.J.W. van der Vegt, and H. Van der Ven. Pseudo-time stepping for space-time discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 219(2):622–643, 2006.

[28] 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. Comput. Phys.*, 217(2):589–611, 2006.

[29] N. Kroll, H. Bieler, H. Deconinck, V. Couallier, H. van der Ven, and K. Sorensen, editors. *ADIGMA A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications*, volume 113 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*. Springer Verlag, 2010.

[30] 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.

[31] G. Vijayasundaram. Transonic flow simulation using upstream centered scheme of Godunov type in finite elements. *J. Comput. Phys.*, 63:416–433, 1986.