Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS

Lukáš Korous

Adaptive hp Discontinuous Galerkin Method for Nonstationary Compressible Euler Equations

Department of Numerical Mathematics

Supervisor: prof. RNDr. Miloslav Feistauer, DrSc., dr. h. c.
Study branch: Numerical and Computational Mathematics

2011

I would like to thank prof. RNDr. Miloslav Feistauer, DrSc., dr. h. c. for his guidance, attentiveness and valuable advice I received from him. Also I would like to thank doc. RNDr. Pavel Šolín, Ph.D for the priceless advices, and enjoyable collaboration on the Finite element software Hermes.

I confirm having prepared the master thesis by my own, and having listed all used sources of information in the bibliography. I agree with lending the master thesis.

In Prague Lukáš Korous

Contents

Název práce: Adaptivní hp nespojitá Galerkinova metoda pro nestacionární stlačitelné Eulerovy rovnice
Autor: Lukáš Korous
Katedra (ústav): Katedra numerické matematiky
Vedoucí bakalárské práce: prof. RNDr. Miloslav Feistauer, DrSc., dr. h. c.
e-mail vedoucího: feist@karlin.mff.cuni.cz

Abstrakt: Stlačitelné Eulerovy rovnice popisují pohyb stlačitelných nevazkých tekutin. Používají se v mnoha oblastech leteckého, automobilového a jaderného inženýrství, chemie, ekologie, klimatologie, i jinde. Matematicky, stlačitelné Eulerovy rovnice představují hyperbolický systém skládající se z několika nelineárních parciálních diferenciálních rovnic (zákony zachování). Tyto rovnice jsou řešeny nejčasteji pomocí metody konečných objemů (MKO), a metody konečných prvků (MKP) nízkého řádu. Nicméně, oba tyto přístupy nedosahují vyššího řádu přesnosti, a navíc je dobře známo, že konformní metoda konečných prvků není optimální nástroj pro diskretizaci rovnic prvního řádu. Nejnadějnější přístup k přibližnému řešení stlačitelných Eulerových rovnic je nespojitá Galerkinova metoda, která kombinuje stabilitu MKO s vynikajícími aproximačními vlastnostmi MKP vyššího řádu. Cílem této diplomové práce byl vývoj, implementace a testování nových algoritmů pro adaptivní řešení nestacionárních stlačitelných Eulerovových rovnic na základě vyššího řádu nespojité Galerkinovy metody (hp-DG). Základem pro nové metody byly nespojitá Galerkinova metoda a časoprostorové hp-MKP algoritmy na dynamických sítích pro nestacionární problémy druhého řádu. Nové algoritmy byly implementovány a testovány v rámci open source knihovny Hermes.

Klíčová slova: numerické simulace, metoda konečných prvků, Eulerovy rovnice, hp-adaptivita, nespojitá Galerkinova metoda

Title: Adaptive hp Discontinuous Galerkin Method for Nonstationary Compressible Euler Equations
Author: Lukáš Korous
Department: Department of Numerical Mathematics
Supervisor: prof. RNDr. Miloslav Feistauer, DrSc., dr. h. c.
Supervisor's e-mail address: feist@karlin.mff.cuni.cz

Abstract: The compressible Euler equations describe the motion of compressible inviscid fluids. They are used in many areas ranging from aerospace, automotive, and nuclear engineering to chemistry, ecology, climatology, and others. Mathematically, the compressible Euler equations represent a hyperbolic system consisting of several nonlinear partial differential equations (conservation laws). These equations are solved most frequently by means of Finite Volume Methods (FVM) and

low-order Finite Element Methods (FEM). However, both these approaches are lacking higher order accuracy and moreover, it is well known that conforming FEM is not the optimal tool for the discretization of first-order equations. The most promissing approach to the approximate solution of the compressible Euler equations is the discontinuous Galerkin method that combines the stability of FVM, with excellent approximation properties of higher-order FEM. The objective of this Master Thesis was to develop, implement and test new adaptive algorithms for the nonstationary compressible Euler equations based on higher-order discontinuous Galerkin (hp-DG) methods. The basis for the new methods will be the discontinuous Galerkin methods and space-time adaptive hp-FEM algorithms on dynamical meshes for nonstationary second-order problems. The new algorithms will be implemented and tested in the framework of the open source library Hermes.

Keywords: numerical simulation, finite element method, Euler equations, hp-adaptivity, discontinuous Galerkin method

Introduction

Discontinuous Galerkin methods (DG methods) in mathematics form a class of numerical methods for solving partial differential equations. They combine features of the finite element and the finite volume framework and have been successfully applied to hyperbolic, elliptic and parabolic problems arising from a wide range of applications. DG methods have in particular received considerable interest for problems with a dominant first-order part, e.g. in electrodynamics, fluid mechanics and plasma physics.

Discontinuous Galerkin methods were first proposed and analyzed in the early 1970s as a technique to numerically solve partial differential equations. In 1973 Reed and Hill introduced a DG method to solve the hyperbolic neutron transport equation.

The origin of the DG method for elliptic problems cannot be traced back to a single publication as features such as jump penalization in the modern sense were developed gradually. However, among the early influential contributors were J. Babuška [2], J.A. Nitsche [16] and M. Zlamal [24]. Interestingly DG methods for elliptic problems were already developed in a paper by Baker in the setting of 4th order equations in 1977. A more complete account of the historical development and an introduction to DG methods for elliptic problems is given in a publication by Arnold, Brezzi, Cockburn and Marini. A number of research directions and challenges on DG methods are collected in the proceedings volume edited by Cockburn, Karniadakis [14] and Shu.

The DG method is very demanding in terms of number of degrees of freedom (DOFs), and thus in terms of the sizes of the linear systems we have to solve. Therefore, techniques to reduce these sizes while keeping sufficiently high quality of the solution of the problems we solve using the DG method are very interesting. The algorithms with the goal to optimize the resolution / cost ratio are known as *adaptive algorithms*. There exist several approaches to adaptively work towards the goal, the core always remains *mesh* or h-adaptivity. For some problems, we can also employ adaptivity in terms of the degree of polynomials we use for approximation of functions that appear in our problems, so called p-adaptivity. The most advanced technique combines these two approaches, and is simply called hp-adaptivity.

Chapter 1

Compressible flow and the Euler equations

1.1 Equations decribing the flow

We consider a time interval $(0, T)$ and space domain $\Omega_t \subset \mathbb{R}^3$ occupied by a fluid at time t. By $\mathcal{M}$ we denote the space-time domain in consideration:

$$\mathcal{M} = \{(\boldsymbol{x}, t)\,; \boldsymbol{x} \in \Omega_t, t \in (0, T)\}\,. \tag{1.1}$$

Moreover we assume that $\mathcal{M}$ is an open set.

1.1.1 Description of the flow

In Computational Fluid Dynamics there exist two classical approaches to the description of the flow, the *Lagrangian description* and the *Eulerian description*.

The idea of the Lagrangian description is to monitor each fluid particle along its pathline (i.e. the curve which the particle traverses in time). If we wanted to set a computational mesh using this description, it would mean to firmly connect nodes of the mesh with certain particles (i.e. the node and the particle would have to share their space coordinates) and move the mesh accordingly to the motion of the fluid as to preserve the node - corresponding particle connection at each time instant. The obvious drawback is the necessity to perform re-meshing operations very frequently, especially when dealing with large distortion of the fluid.

The Eulerian description focuses on fluid particles that move through fixed points within a computational domain. In other words, whereas in the Lagrangian description the particle was fixed and the point in space it was currently occupying was changing, now it is the point in space that holds still and the particle in consideration is changing and is always corresponding to the one that is currently occupying

the considered point in space. From this idea it follows that a computational mesh for this description would be fixed with respect to time.

It is not difficult to imagine that formulation of some basic mechanical principles could be easier for the moving particle, that is using the Lagrangian approach. However, the Eulerian description is used for the formulation of conservation laws as will be seen in the following subsections. Last, using this approach, large distortions of fluid domain can be handled with relative ease.

We shall proceed now with basics of the Lagrangian and the Eulerian descriptions and their relation. We shall present the equations describing the flow derived from conservation laws in their integral forms using the Eulerian approach.

Lagrangian description
We specify the particle in consideration using the mapping

$$\boldsymbol{\varphi}\left(\boldsymbol{X}, t_0; t\right) \tag{1.2}$$

which determines the current (at time t) position $\boldsymbol{x} \in \Omega_t$ of the particle that occupies the point $\boldsymbol{X}$ at time t_0, i.e.

$$\boldsymbol{x} = \boldsymbol{\varphi}\left(\boldsymbol{X}, t_0; t\right), \quad \boldsymbol{X} \in \Omega_{t_0}, \tag{1.3}$$

where we can omit the reference time t_0 and write

$$\boldsymbol{x} = \boldsymbol{\varphi}\left(\boldsymbol{X}, t\right). \tag{1.4}$$

Customarily the components X_1, X_2, X_3 of the reference point $\boldsymbol{X}$ are called the *Lagrangian coordinates* and the components x_1, x_2, x_3 of the point $\boldsymbol{x}$ in the current configuration Ω_t are called the *Eulerian coordinates*. The *velocity* and *acceleration* of the particle given by the reference point $\boldsymbol{X}$ are defined as

$$\hat{v}\left(\boldsymbol{X}, t\right) = \frac{\partial \boldsymbol{\varphi}}{\partial t}\left(\boldsymbol{X}, t_0; t\right), \tag{1.5}$$

$$\hat{a}\left(\boldsymbol{X}, t\right) = \frac{\partial^2 \boldsymbol{\varphi}}{\partial t^2}\left(\boldsymbol{X}, t_0; t\right), \tag{1.6}$$

provided the above derivatives exist.

Eulerian description
Using once again the mapping $\boldsymbol{\varphi}$ defined in (1.2), relation (1.4) and the Lagrangian definition of velocity (1.5), we can express the *velocity* of the fluid particle passing through the point $\boldsymbol{x}$ at time t:

$$\boldsymbol{v}\left(\boldsymbol{x}, t\right) = \hat{\boldsymbol{v}}\left(\boldsymbol{X}, t\right) = \frac{\partial \boldsymbol{\varphi}}{\partial t}\left(\boldsymbol{X}, t\right), \tag{1.7}$$

where $\boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X}, t)$.
We shall demand the following regularity from the velocity function:

$$\boldsymbol{v} \in \left[\mathcal{C}^1(\mathcal{M})\right]^3. \tag{1.8}$$

We can pass to the Eulerian coordinates from the Lagrangian ones by solving the following initial value problem:

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{v}(\boldsymbol{x}, t), \quad \boldsymbol{x}(t_0) = \boldsymbol{X}. \tag{1.9}$$

Under assumption (1.8), the problem (1.9) has exactly one maximal solution $\boldsymbol{\varphi}(\boldsymbol{X}, t_0, t)$ for each $(\boldsymbol{X}, t_0) \in \mathcal{M}$ defined for t from a certain subinterval of $(0, T)$. Moreover, in its domain of definition, the mapping $\boldsymbol{\varphi}$ has continuous first order derivatives with respect to $X_1,\ X_2,\ X_3,\ t_0,\ t$ and continuous second order derivatives $\partial^2\boldsymbol{\varphi}/\partial t \partial X_i$, $\partial^2\boldsymbol{\varphi}/\partial t_0 \partial X_i$, $i = 1,\ 2,\ 3$. These statements result from theory of classical solutions of ordinary differential equations.

Under assumption (1.8), the *acceleration* of the particle passing through the point $\boldsymbol{x}$ at time t can be expressed as

$$\boldsymbol{a}(\boldsymbol{x}, t) = \frac{\partial \boldsymbol{v}}{\partial t}(\boldsymbol{x}, t) + \sum_{i=1}^{3} v_i(\boldsymbol{x}, t) \frac{\partial \boldsymbol{v}}{\partial x_i}(\boldsymbol{x}, t). \tag{1.10}$$

This, written in a short form reads

$$\boldsymbol{a} = \frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \mathsf{grad})\, \boldsymbol{v} = \frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\, \boldsymbol{v}, \tag{1.11}$$

where the differentiation represented by the symbol ∇ is with respect to the spatial variables x_1, x_2, x_3.

1.1.2 The Transport theorem

We wish to study some physical quantity which is transported by fluid particles in our space-time domain $\mathcal{M}$. Let a function $F = F(\boldsymbol{x}, t) : \mathcal{M} \longrightarrow \mathbb{R}$ represent some physical quantity in the Eulerian coordinates, and let us consider a system of fluid particles filling a bounded domain $\mathcal{V}(t) \subset \Omega_t$ at time t. By $\mathcal{F}$ we denote the total amount of the quantity represented by the function F contained in $\mathcal{V}(t)$:

$$\mathcal{F}(t) = \int_{\mathcal{V}(t)} F(\boldsymbol{x}, t)\, d\boldsymbol{x}. \tag{1.12}$$

For the formulation of fundamental equations describing the flow we need to calculate the rate of change of the quantity $\mathcal{F}$ bound on the system of particles considered. In other words we shall be interested in the derivative

$$\frac{d\mathcal{F}(t)}{dt} = \frac{d}{dt} \int_{\mathcal{V}(t)} F(\boldsymbol{x}, t)\, d\boldsymbol{x}. \tag{1.13}$$

In the following we shall suppose that (1.8) holds.

Lemma 1.1
Let $t_0 \in (0,T)\,, \mathcal{V}(t_0)$ be a bounded domain and let $\overline{\mathcal{V}(t_0)} \subset \Omega_{t_0}$. Then there exist an interval $(t_1,t_2) \subset (0,T)$, $t_0 \in (t_1,t_2)$ such that the following conditions are satisfied:

a) *The mapping '$t \in (t_1,t_2)\,, \boldsymbol{X} \in \mathcal{V}(t_0) \longrightarrow \boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X},t_0;t) \in \mathcal{V}(t)$' has continuous first order derivatives with respect to t, X_1, X_2, X_3 and continuous second order derivatives $\partial^2\boldsymbol{\varphi}/\partial t \partial X_i,\ i = 1,2,3$.*

b) *The mapping '$\boldsymbol{X} \in \mathcal{V}(t_0) \longrightarrow \boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X},t_0;t) \in \mathcal{V}(t)$' is for all $t \in (t_1,t_2)$ a continuously differentiable one-to-one mapping of $\mathcal{V}(t_0)$ onto $\mathcal{V}(t)$ with continuous and bounded Jacobian $\mathcal{J}(\boldsymbol{X},t)$ which satisfies the condition*

$$\mathcal{J}(\boldsymbol{X},t) > 0 \quad \forall \boldsymbol{X} \in \mathcal{V}(t_0)\,,\ \forall t \in (t_1,t_2)\,.$$

c) *The inclusion*

$$\left\{(\boldsymbol{x},t)\,;\ t \in [t_1,t_2]\,, \boldsymbol{x} \in \overline{\mathcal{V}(t)}\right\} \subset \mathcal{M}$$

holds and therefore the mapping $\boldsymbol{v}$ has continuous and bounded first order derivatives on $\{(\boldsymbol{x},t)\,;\ t \in (t_1,t_2)\,, \boldsymbol{x} \in \mathcal{V}(t)\}$ with respect to all variables.

d) $\boldsymbol{v}(\boldsymbol{\varphi}(\boldsymbol{X},t_0;t),t) = \frac{\partial \boldsymbol{\varphi}}{\partial t}(\boldsymbol{X},t_0;t)\ \ \forall \boldsymbol{X} \in \mathcal{V}(t_0)\,,\ \forall\ t \in (t_1,t_2)\,.$

For proof, see [8].

Theorem 1.2 - The transport theorem
Let conditions from Lemma 1.1, a)-d) be satisfied and let the function $F = F(\boldsymbol{x},t)$ have continuous and bounded first order derivatives on the set $\{(\boldsymbol{x},t)\,; t \in (t_1,t_2)\,,\ x \in \mathcal{V}(t)\}$. Let $\mathcal{F}(t) = \int_{\mathcal{V}(t)} F(\boldsymbol{x},t)\,d\boldsymbol{x}$.
Then for each $t \in (t_1,t_2)$ there exists a finite derivative

$$\begin{aligned}\frac{d\mathcal{F}}{dt}(t) &= \frac{d}{dt}\int_{\mathcal{V}(t)} F(\boldsymbol{x},t)\,d\boldsymbol{x} \\ &= \int_{\mathcal{V}(t)} \left[\frac{\partial F}{\partial t}(\boldsymbol{x},t) + \operatorname{div}(F\boldsymbol{v})(\boldsymbol{x},t)\right] d\boldsymbol{x}. \qquad (1.14)\end{aligned}$$

For proof, see [9].

1.1.3 The continuity equation

The *density of fluid is* a function

$$\rho : \mathcal{M} \longrightarrow (0,+\infty)$$

which allows us to determine the mass $m(\mathcal{V};t)$ of the fluid contained in any subdomain $\mathcal{V} \subset \Omega_t$:

$$m(\mathcal{V};t) = \int_{\mathcal{V}} \rho(\boldsymbol{x},t)\, d\boldsymbol{x}. \tag{1.15}$$

Assumptions 1.3
In what follows, let $\rho \in \mathcal{C}^1(\mathcal{M})$ and as before let $\boldsymbol{v} \in [\mathcal{C}^1(\mathcal{M})]^3$. We shall consider an arbitrary time instant $t_0 \in (0,T)$ and a moving piece of fluid formed by the same particles at each time instant and filling at time t_0 a bounded domain $\mathcal{V} \subset \overline{\mathcal{V}} \subset \Omega_{t_0}$ with a Lipschitz-continuous boundary $\partial\mathcal{V}$ called the *control volume* in the domain Ω_{t_0}. By $\mathcal{V}(t)$ we denote the domain occupied by this piece of fluid at time $t \in (t_1,t_2)$, where (t_1,t_2) is a sufficiently small interval containing t_0 with properties from Lemma 1.1.

Since the domain $\mathcal{V}(t)$ is formed by the same particles at each time instant, the *conservation of mass* can be formulated in the following way: *The mass of the piece of fluid represented by the domain $\mathcal{V}(t)$ does not depend on time t.* This means that

$$\frac{dm(\mathcal{V}(t);t)}{dt} = 0,\ t \in (t_1,t_2), \tag{1.16}$$

where with respect to (1.15) we have

$$m(\mathcal{V}(t);t) = \int_{\mathcal{V}(t)} \rho(\boldsymbol{x},t)\, d\boldsymbol{x}. \tag{1.17}$$

From Theorem 1.2 for a function $F := \rho$ we get the identity

$$\int_{\mathcal{V}(t)} \left[\frac{\partial\rho}{\partial t}(\boldsymbol{x},t) + \operatorname{div}(\rho\boldsymbol{v})(\boldsymbol{x},t)\right] d\boldsymbol{x} = 0, \quad t \in (t_1,t_2). \tag{1.18}$$

If we substitute $t := t_0$ and take into account that $\mathcal{V}(t_0) = \mathcal{V}$, we conclude that

$$\int_{\mathcal{V}} \left[\frac{\partial\rho}{\partial t}(\boldsymbol{x},t_0) + \operatorname{div}(\rho\boldsymbol{v})(\boldsymbol{x},t_0)\right] d\boldsymbol{x} = 0 \tag{1.19}$$

for an arbitrary $t_0 \in (0,T)$ and an arbitrary control volume $\mathcal{V} \subset \Omega_{t_0}$. We use the following Lemma in order to derive the differential form of the law of conservation of mass:

Lemma 1.4
Let $\Omega \subset \mathbb{R}^N$ be an open set and let $f \in \mathcal{C}(\Omega)$. Then the following holds:

$f \equiv 0$ in Ω if and only if $\int_{\mathcal{V}} f(\boldsymbol{x})\, d\boldsymbol{x} = 0$ for any bounded open set $\mathcal{V} \subset \overline{\mathcal{V}} \subset \Omega$.

Now we use Lemma 1.4 and obtain the differential form of the law of conservation of mass called the *continuity equation*:

$$\frac{\partial \rho}{\partial t}(\boldsymbol{x},t) + \operatorname{div}(\rho(\boldsymbol{x},t)\,\boldsymbol{v}(\boldsymbol{x},t)) = 0, \quad \boldsymbol{x} \in \Omega_t,\ t \in (0,T). \tag{1.20}$$

1.1.4 The equations of motion

We proceed by deriving basic dynamical equations describing flow motion from the *law of conservation of momentum* which can be formulated in this way:

The rate of change of the total momentum of a piece of fluid formed by the same particles at each time and occupying the domain $\mathcal{V}(t)$ at time instant t is equal to the force acting on $\mathcal{V}(t)$.

Let assumptions 1.3 be satisfied. The total momentum of particles contained in $\mathcal{V}(t)$ is given by

$$\boldsymbol{\mathcal{H}}(\mathcal{V}(t)) = \int_{\mathcal{V}(t)} \rho(\boldsymbol{x},t)\,\boldsymbol{v}(\boldsymbol{x},t)\,d\boldsymbol{x}. \tag{1.21}$$

Moreover, denoting by $\boldsymbol{\mathcal{F}}(\mathcal{V}(t))$ the force acting on the volume $\mathcal{V}$, the law of conservation of momentum reads

$$\frac{d\boldsymbol{\mathcal{H}}(\mathcal{V}(t))}{dt} = \boldsymbol{\mathcal{F}}(\mathcal{V}(t)), \quad t \in (t_1,t_2). \tag{1.22}$$

Using Theorem 1.2 for functions $F := \rho v_i$, $i = 1,2,3$, we get

$$\int_{\mathcal{V}(t)} \left[\frac{\partial}{\partial t}(\rho(\boldsymbol{x},t)\,v_i(\boldsymbol{x},t)) + \operatorname{div}(\rho(\boldsymbol{x},t)\,v_i(\boldsymbol{x},t)\,\boldsymbol{v}(\boldsymbol{x},t))\right] d\boldsymbol{x} = \mathcal{F}_i(\mathcal{V}(t)), \tag{1.23}$$

$$i = 1,2,3,\ t \in (t_1,t_2).$$

Taking into account that $t_0 \in (0,T)$ is an arbitrary time instant and $\mathcal{V}_{t_0} = \mathcal{V} \subset \overline{\mathcal{V}} \subset \Omega_{t_0}$, where $\mathcal{V}$ is an arbitrary control volume, we get the law of conservation of momentum in the form where we write t instead of t_0:

$$\int_{\mathcal{V}} \left[\frac{\partial}{\partial t}(\rho(\boldsymbol{x},t)\,v_i(\boldsymbol{x},t)) + \operatorname{div}(\rho(\boldsymbol{x},t)\,v_i(\boldsymbol{x},t)\,\boldsymbol{v}(\boldsymbol{x},t))\right] d\boldsymbol{x} = \mathcal{F}_i(\mathcal{V};t), \tag{1.24}$$

$i = 1, 2, 3$, for an arbitrary $t \in (0,T)$ and an arbitrary control volume $\mathcal{V} \subset \overline{\mathcal{V}} \subset \Omega_t$.

According to [8], the components $\mathcal{F}_i(\mathcal{V};t)$, $i=1,2,3$, of the vector $\boldsymbol{\mathcal{F}}(\mathcal{V};t)$ can be expressed as

$$\mathcal{F}_i(\mathcal{V};t)=\int_{\mathcal{V}}\rho(\boldsymbol{x},t)\,f_i(\boldsymbol{x},t)\,dx+\int_{\partial\mathcal{V}}\sum_{j=1}^{3}\tau_{ji}(\boldsymbol{x},t)\,n_j(\boldsymbol{x})\,dS,\ i=1,2,3, \tag{1.25}$$

assuming that $\tau_{ij}\in\mathcal{C}^1(\mathcal{M})$ and $f_i\in\mathcal{C}(\mathcal{M})$, $(i,\,j=1,2,3)$. Here τ_{ji} are components of the *stress tensor* $\mathcal{T}$ and f_i are components of the *density of the volume force* $\boldsymbol{f}$. Substituting this into (1.24), we get

$$\int_{\mathcal{V}}\left[\frac{\partial}{\partial t}\left(\rho(\boldsymbol{x},t)\,v_i(\boldsymbol{x},t)\right)+\operatorname{div}\left(\rho(\boldsymbol{x},t)\,v_i(\boldsymbol{x},t)\,\boldsymbol{v}(\boldsymbol{x},t)\right)\right]dx= \tag{1.26}$$

$$\int_{\mathcal{V}}\rho(\boldsymbol{x},t)\,f_i(\boldsymbol{x},t)\,dx+\int_{\partial\mathcal{V}}\sum_{j=1}^{3}\tau_{ji}(\boldsymbol{x},t)\,n_j(\boldsymbol{x})\,dS,\quad i=1,2,3, \tag{1.27}$$

for each $t\in(0,T)$ and an arbitrary control volume $\mathcal{V}$ in Ω_t. Moreover, applying Green's theorem and Lemma 1.4, we obtain the desired *equation of motion of a general fluid in the differential conservative form*

$$\frac{\partial}{\partial t}(\rho v_i)+\operatorname{div}(\rho v_i\boldsymbol{v})=\rho f_i+\sum_{j=1}^{3}\frac{\partial\tau_{ji}}{\partial x_j},\quad i=1,2,3. \tag{1.28}$$

This can be written as

$$\frac{\partial}{\partial t}(\rho\boldsymbol{v})+\operatorname{div}(\rho\boldsymbol{v}\otimes\boldsymbol{v})=\rho\boldsymbol{f}+\operatorname{div}\mathcal{T}, \tag{1.29}$$

where $\otimes$ denotes the *tensor product*:

$$\boldsymbol{a}\otimes\boldsymbol{b}=\begin{pmatrix}a_1b_1 & a_1b_2 & a_1b_3\\ a_2b_1 & a_2b_2 & a_2b_3\\ a_3b_1 & a_3b_2 & a_3b_3\end{pmatrix},$$

and $\operatorname{div}(\boldsymbol{a}\otimes\boldsymbol{b})$ is a vector quantity:

$$\operatorname{div}(\boldsymbol{a}\otimes\boldsymbol{b})=\left(\sum_{i=1}^{3}\frac{\partial}{\partial x_i}a_ib_1,\ \sum_{i=1}^{3}\frac{\partial}{\partial x_i}a_ib_2,\ \sum_{i=1}^{3}\frac{\partial}{\partial x_i}a_ib_3\right)^T.$$

1.1.5 The Navier-Stokes equations

The relation between the stress tensor and other quantities describing fluid flow, the velocity and its derivatives in particular, represent the so-called *rheological*

equations of the fluid. For the derivation of the Navier-Stokes equations we shall use

$$\mathcal{T} = (-p + \lambda \mathrm{div}\, \boldsymbol{v})\, \mathbb{I} + 2\mu \mathbb{D}(\boldsymbol{v}), \tag{1.30}$$

where $\mathbb{D}$ is the deformation velocity tensor:

$$\mathbb{D} = \mathbb{D}(\boldsymbol{v}) = (d_{ij})_{i,j=1}^{3}, \; d_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right), \tag{1.31}$$

λ, μ are constants or scalar functions of thermodynamical quantities, λ and μ are called the *first* and the *second viscosity coefficient* respectively. For the assumptions under which we can write (1.30) see [8]. Altough viscosity coefficients can be functions of thermodynamical quantities (most important of which is θ, the absolute temperature) we shall treat them as if they were constants. Let assumptions 1.3 be satisfied and let us assume that

$$\frac{\partial \boldsymbol{v}}{\partial t} \in [\mathcal{C}(\mathcal{M})]^3, \; \frac{\partial^2 \boldsymbol{v}}{\partial x_i \partial x_j} \in [\mathcal{C}(\mathcal{M})]^3 \;\; (i, j = 1, 2, 3). \tag{1.32}$$

Now let us substitute relation (1.30) into the general equations of motion (1.29) with the assumption of constant viscosity coefficients and assumptions (1.32). We come to the Navier-Stokes equations in the form

$$\frac{\partial(\rho \boldsymbol{v})}{\partial t} + \mathrm{div}\,(\rho \boldsymbol{v} \otimes \boldsymbol{v}) = \rho \boldsymbol{f} - \nabla p + \mu \triangle \boldsymbol{v} + (\mu + \lambda)\, \nabla \mathrm{div}\, \boldsymbol{v}. \tag{1.33}$$

For details see [9].

1.1.6 The energy equation

Now let us derive a differential equation equivalent to the *law of conservation of energy*. As in the preceding subsections, we consider a piece of fluid represented by a control volume $\mathcal{V}(t)$ satisfying assumptions 1.3. The law of conservation of energy can be formulated as follows:

The rate of change of the total energy of the fluid particles, occupying the domain $\mathcal{V}(t)$ at time t, is equal to the sum of powers of the volume force acting on the volume $\mathcal{V}(t)$ and the surface force acting on the surface $\partial\mathcal{V}(t)$, and of the amount of heat transmitted to $\mathcal{V}(t)$.

By $\mathcal{E}(\mathcal{V}(t))$ let us denote the total energy of the fluid particles contained in the domain $\mathcal{V}(t)$ and by $\mathcal{Q}(\mathcal{V}(t))$ the amount of heat transmitted to $\mathcal{V}(t)$ at time t. Taking into account the character of volume and surface forces involved, we get the identity representing the law of conservation of energy:

$$\frac{d}{dt}\mathcal{E}(\mathcal{V}(t)) = \int_{\mathcal{V}(t)} \rho(\boldsymbol{x}, t)\, \boldsymbol{f}(\boldsymbol{x}, t) \cdot \boldsymbol{v}(\boldsymbol{x}, t)\, d\boldsymbol{x} \tag{1.34}$$

$$+ \int_{\partial\mathcal{V}(t)} \sum_{i,j=1}^{3} \tau_{ji}(\boldsymbol{x}, t)\; n_j(\boldsymbol{x})\; v_i(\boldsymbol{x}, t)\, dS + \mathcal{Q}(\mathcal{V}(t)).$$

Following relations hold:

$$a) \quad \mathcal{E}(\mathcal{V}(t)) = \int_{\mathcal{V}(t)} E(\boldsymbol{x},t)\, d\boldsymbol{x}, \tag{1.35}$$

$$b) \quad E = \rho\left(e + \frac{|\boldsymbol{v}|^2}{2}\right),$$

$$c) \quad \mathcal{Q}(\mathcal{V}(t)) = \int_{\mathcal{V}(t)} \rho(\boldsymbol{x},t)\, q(\boldsymbol{x},t)\, d\boldsymbol{x} - \int_{\partial\mathcal{V}(t)} \boldsymbol{\phi}_q(\boldsymbol{x},t)\cdot \boldsymbol{n}(\boldsymbol{x})\, dS.$$

Here E is the total energy, e is the density of the specific internal energy (related to the unit mass) associated with molecular and atomic behavior, $|\boldsymbol{v}|^2/2$ is the density of the kinetic energy, q represents the density of heat sources (again related to the unit mass) and $\boldsymbol{\phi_q}$ is the heat flux.

Let assumptions 1.3 hold and further let $\tau_{ij}, \left(\boldsymbol{\phi}_q\right)_i \in \mathcal{C}^1(\mathcal{M})$ and $f_i, q \in \mathcal{C}(\mathcal{M})$ $(i,j=1,2,3)$. Using this, relations (1.35) *a)-c)*, Theorem 1.2, Green's theorem and Lemma 1.4, we derive from (1.34) the differential *energy equation*, where we take advantage of (1.30):

$$\frac{\partial E}{\partial t} + \operatorname{div}(E\boldsymbol{v}) = \rho\boldsymbol{f}\cdot\boldsymbol{v} - \operatorname{div}(p\boldsymbol{v}) + \operatorname{div}(\lambda\boldsymbol{v}\operatorname{div}\boldsymbol{v}) + \operatorname{div}(2\mu\mathbb{D}(\boldsymbol{v})\,\boldsymbol{v}) + \rho q - \operatorname{div}\boldsymbol{\phi}_q. \tag{1.36}$$

For details see [9].

1.1.7 Thermodynamical relations

In order to complete the equations describing the flow, some other relations shall be added. The system now contains seven unknown quantities: $v_1, v_2, v_3, \rho, e, \theta, p$, but only 5 equations (scalar continuity equation, vector Navier-Stokes equations and scalar energy equation), i.e. $(1+3+1)=5$. From this we see, that additional two equations should be included.

Basic Thermodynamical Quantities
The absolute temperature θ, the density ρ and the pressure p are called the *state variables*. All these quantities are positive scalar functions. We consider only the so-called *perfect gas* or *ideal gas* whose state variables satisfy the following *equation of state*

$$p = R\theta\rho, \tag{1.37}$$

where R is the *gas constant*, which is defined as

$$R = c_p - c_v. \tag{1.38}$$

Here c_p denotes the *specific heat at constant pressure*, i.e. the ratio of the increment of the amount of heat related to the unit mass, to the increment of temperature at constant pressure. Analogously c_v denotes the *specific heat at constant volume*. Experiments show that $c_p > c_v$, so that $R > 0$, and that c_p and c_v can be treated like constants for a relatively large range of temperature. We set $\gamma = c_p/c_v$ which is the so-called *Poisson adiabatic constant*. The internal energy related to the unit mass is defined by

$$e = c_v\theta, \tag{1.39}$$

which explains the meaning of the internal energy: it is the amount of heat it would have to be transmitted out of the fluid so that its temperature would reach (absolute) zero, volume being kept constant during the whole process.

With respect to the above relations, we can express the internal energy as

$$e = c_p\theta - R\theta. \tag{1.40}$$

The complete system of equations describing the flow
The complete system now reads

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \boldsymbol{v}) &= 0, && (1.41)\\
\frac{\partial(\rho \boldsymbol{v})}{\partial t} + \operatorname{div}(\rho \boldsymbol{v} \otimes \boldsymbol{v}) &= \rho \boldsymbol{f} - \nabla p + \mu \triangle \boldsymbol{v} + (\mu + \lambda)\, \nabla \operatorname{div} \boldsymbol{v}, && (1.42)\\
\frac{\partial E}{\partial t} + \operatorname{div}(E \boldsymbol{v}) &= \rho \boldsymbol{f} \cdot \boldsymbol{v} - \operatorname{div}(p\boldsymbol{v}) + \operatorname{div}(\lambda \boldsymbol{v} \operatorname{div} \boldsymbol{v}) + && (1.43)\\
&+ \operatorname{div}(2\mu \mathbb{D}(\boldsymbol{v})\, \boldsymbol{v}) + \rho q - \operatorname{div} \boldsymbol{\phi}_q,\\
p &= (\gamma - 1)\left(E - \rho |\boldsymbol{v}|^2/2\right), && (1.44)\\
\theta &= \left(E/\rho - |\boldsymbol{v}|^2/2\right)/c_v. && (1.45)
\end{aligned}$$

This system is simply called the *compressible Navier-Stokes equations* for a heat-conductive perfect gas. Equations (1.44) and (1.45) follow from (1.37) - (1.40) and (1.35).

1.1.8 Entropy and the second law of thermodynamics

Entropy One of the important thermodynamical quantities is the entropy S, defined by the relation

$$\theta dS = de + pdV, \tag{1.46}$$

where $V = 1/\rho$ is the so-called specific volume. This identity is derived in thermodynamics under the assumption that the internal energy is a function of S and V, $e = e(S, V)$, which explains the meaning of the differentials in (1.46).

Theorem 1.3
For a perfect gas we have

$$S = c_v \ln \frac{p/p_0}{(\rho/\rho_0)^\gamma} + const \tag{1.47}$$

$$= c_v \ln \frac{\theta/\theta_0}{(\rho/\rho_0)^{\gamma-1}} + const, \tag{1.48}$$

where p_0 and ρ_0 are fixed (reference) values of pressure and density and $\theta_0 = p_0/(R\rho_0)$.

Proof
Using (1.39) *and the relation* $V = 1/\rho$, *we can write* (1.46) *in the form*

$$\theta dS = c_v d\theta - \frac{p d\rho}{\rho^2}. \tag{1.49}$$

From this and (1.37) - (1.39) *we get*

$$dS = c_v \frac{d\theta}{\theta} - \frac{p}{\rho\theta}\frac{d\rho}{\rho} = c_v \frac{d(p/\rho)}{(p/\rho)} - R\frac{d\rho}{\rho} = c_v d \ln \frac{p/p_0}{(\rho/\rho_0)^\gamma} = c_v d \ln \frac{\theta/\theta_0}{(\rho/\rho_0)^{\gamma-1}}, \tag{1.50}$$

which immediately yields (1.47).

The second law of thermodynamics In the irreversible processes, equality (1.47) does not hold in general and is replaced by the inequality

$$dS \geq \frac{\delta \mathcal{Q}}{\theta} \tag{1.51}$$

called the *second law of thermodynamics*. For a system of fluid particles occupying a domain$\mathcal{V}(t)$ at time t we postulate the second law of thermodynamics mathematically in the form

$$\frac{d}{dt}\int_{\mathcal{V}(t)} \rho(\boldsymbol{x},t)\, S(\boldsymbol{x},t)\, d\boldsymbol{x} \geq \int_{\mathcal{V}(t)} \frac{\rho(\boldsymbol{x},t)\, q(\boldsymbol{x},t)}{\theta(\boldsymbol{x},t)} d\boldsymbol{x} - \int_{\delta\mathcal{V}(t)} \frac{\boldsymbol{\phi}_q(\boldsymbol{x},t)\cdot \boldsymbol{n}(\boldsymbol{x})}{\theta(\boldsymbol{x},t)} dS. \tag{1.52}$$

The left-hand side of (1.52) represents the rate of change of the entropy contained in the volume $\mathcal{V}(t)$, and the first and second integral on the right-hand side are called the *entropy production* and the *entropy flux*. Let ρ, θ, v_i, $\boldsymbol{\phi}_{q_i} \in \mathcal{C}^1(\mathcal{M})$; $q, f_i \in \mathcal{C}(\mathcal{M})$, $i = 1,2,3$. By virtue of the transport Theorem 1.2 and the continuity equation (1.20), from (1.52) we obtain the inequality

$$\rho\frac{\partial(\rho S)}{\partial t} + \operatorname{div}(\rho S \boldsymbol{v}) \geq \frac{\rho q}{\theta} - \operatorname{div}\left(\frac{\boldsymbol{\phi}_q}{\theta}\right). \tag{1.53}$$

A more general model of flow is obtained in thermodynamics under the assumption that the pressure is a function of the density and entropy: $p = p(\rho, S)$, where p is a continuously differentiable function and $\partial p/\partial \rho > 0$. Let us introduce the quantity

$$a = \sqrt{\frac{\partial p}{\partial \rho}} \tag{1.54}$$

which has the dimension ms^{-1} of velocity and is called the *speed of sound*. Another important characteristic of the flow is so-called *Mach number*, which is defined as

$$M = \frac{|\boldsymbol{v}|}{a} \ , \tag{1.55}$$

where $\boldsymbol{v}$ is the flow velocity.

1.2 Euler equations and their properties

If we set $\mu = \lambda = k = 0$, we obtain the model of inviscid compressible flow, described by the continuity equation, the Euler equations, the energy equation and thermodynamical relations. Since gases are light, usually it is possible to neglect the effect of the volume force. Neglecting heat sources too (assuming *adiabaticflow*), we get the system for the perfect inviscid gas in the following form:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \boldsymbol{v}) &= 0, && (1.56)\\ \frac{\partial(\rho \boldsymbol{v})}{\partial t} + \operatorname{div}(\rho \boldsymbol{v} \otimes \boldsymbol{v}) + \nabla p &= 0, && (1.57)\\ \frac{\partial E}{\partial t} + \operatorname{div}((E+p)\,\boldsymbol{v}) &= 0, && (1.58)\\ p &= (\gamma - 1)\left(E - \rho\,|\boldsymbol{v}|^2/2\right). && (1.59) \end{aligned}$$

This system is simply called the *compressible Euler equations*.

In the following we shall be concerned with the 2-dimensional case, which will be used in numerical examples. All results apply for the 3-dimensional case as well.

1.2.1 Dimensionless Euler equations

We set 3 similarity constants l_r, v_r, and ρ_r, which are, in order: characteristic length, characteristic velocity, and characteristic density. Now we multiply the Euler equations by these constants in the following way:

$$\begin{aligned} \left[\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \boldsymbol{v})\right] \frac{l_r}{\rho_r v_r} &= 0, && (1.60)\\ \left[\frac{\partial(\rho \boldsymbol{v})}{\partial t} + \operatorname{div}(\rho \boldsymbol{v} \otimes \boldsymbol{v}) + \nabla p\right] \frac{l_r}{\rho_r v_r^2} &= 0, && (1.61)\\ \left[\frac{\partial E}{\partial t} + \operatorname{div}((E+p)\boldsymbol{v})\right] \frac{l_r}{\rho_r v_r^3} &= 0. && (1.62) \end{aligned}$$

Then we get the dimensionless Euler equations in the form

$$\begin{aligned} \frac{\partial \tilde{\rho}}{\partial \tilde{t}} + \tilde{\nabla} \cdot (\tilde{\rho} \tilde{\boldsymbol{v}}) &= 0, && (1.63)\\ \frac{\partial(\tilde{\rho} \tilde{\boldsymbol{v}})}{\partial \tilde{t}} + \tilde{\nabla} \cdot (\tilde{\rho} \tilde{\boldsymbol{v}} \otimes \tilde{\boldsymbol{v}}) + \tilde{\nabla} \tilde{p} &= 0, && (1.64)\\ \frac{\partial \tilde{E}}{\partial \tilde{t}} + \tilde{\nabla} \cdot ((\tilde{E} + \tilde{p}) \tilde{\boldsymbol{v}}) &= 0, && (1.65) \end{aligned}$$

where

$$t_r = \frac{l_r}{v_r}, \tag{1.66}$$

$$\tilde{t} = \frac{t}{t_r}, \tag{1.67}$$

$$\tilde{\rho} = \frac{\rho}{\rho_r}, \tag{1.68}$$

$$\tilde{\boldsymbol{v}} = \frac{\boldsymbol{v}}{v_r}, \tag{1.69}$$

$$\tilde{\nabla} = l_r \nabla, \tag{1.70}$$

$$\tilde{E} = \frac{E}{\rho_r u_r^2}, \tag{1.71}$$

$$\tilde{p} = \frac{p}{\rho_r u_r^2}. \tag{1.72}$$

As we see, the dimensionless Euler equations have the same form as the original equations, only the relations are rescaled using the above relations. Therefore, in the sequel, we shall omit the symbol '˜' over the considered quantities.

1.2.2 Conservative form of the Euler equations

The system of governing equations can be written in the so-called *conservative form*.

$$\frac{\partial \boldsymbol{w}}{\partial t} + \frac{\partial \boldsymbol{f}_1}{\partial x_1} + \frac{\partial \boldsymbol{f}_2}{\partial x_2} = 0, \tag{1.73}$$

where

$$\boldsymbol{w} = \begin{pmatrix} \varrho \\ \rho v_1 \\ \rho v_2 \\ E \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}, \tag{1.74}$$

$$\boldsymbol{f}_1 = \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ v_1(E+p) \end{pmatrix} = \begin{pmatrix} w_2 \\ \frac{w_2^2}{w_1} + p \\ \frac{w_2 w_3}{w_1} \\ \frac{w_2}{w_1}(w_4 + p) \end{pmatrix}, \tag{1.75}$$

$$\boldsymbol{f}_2 = \begin{pmatrix} \rho v_2 \\ \rho v_2 v_1 \\ \rho v_2^2 + p \\ v_2(E+p) \end{pmatrix} = \begin{pmatrix} w_2 \\ \frac{w_3 w_2}{w_1} \\ \frac{w_3^2}{w_1} + p \\ \frac{w_3}{w_1}(w_4 + p) \end{pmatrix}, \tag{1.76}$$

$$p = (\gamma - 1)\left(w_4 - \frac{w_2^2 + w_3^2}{2w_1}\right). \tag{1.77}$$

Usually, $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$, are called *inviscid Euler fluxes*. The unknowns ρ, v_1, v_2, p in the original system of equations are called *primitive variables*, whereas $w_1 = \rho, w_2 = \rho v_1, w_3 = \rho v_2, w_4 = E$ are called *conservative variables*.

The domain of definition of the vector-valued functions $\boldsymbol{f}_1, \boldsymbol{f}_2$ is the open set $D \subset \mathrm{R}^2$ where the corresponding density and pressure are positive:

$$D = \left\{ \boldsymbol{w} \in \mathrm{R}^4;\ w_1 = \rho > 0, w_4 - \frac{w_2^2 + w_3^2}{2w_1} = \frac{p}{\gamma - 1} > 0 \right\}. \tag{1.78}$$

One can see that the inviscid Euler fluxes are continuously differentiable in their domain of definition. Differentiation in (1.73) and the chain rule lead to a first order system of quasilinear partial differential equations

$$\frac{\partial \boldsymbol{w}}{\partial t} + \mathbb{A}_1(\boldsymbol{w}) \frac{\partial \boldsymbol{w}}{\partial x_1} + \mathbb{A}_2(\boldsymbol{w}) \frac{\partial \boldsymbol{w}}{\partial x_2} = 0, \tag{1.79}$$

where $\mathbb{A}_s(\boldsymbol{w}), s = 1, 2$ are $m \times m$ matrices called *flux Jacobians* defined as the Jacobi matrices of the mappings $\boldsymbol{f}_s, s = 1, 2$, at a point $\boldsymbol{w} \in D$:

$$\mathbb{A}_s(\boldsymbol{w}) = \frac{D\boldsymbol{f}_s(\boldsymbol{w})}{D\boldsymbol{w}} = \left(\frac{\partial f_{si}(\boldsymbol{w})}{\partial w_j} \right)_{i,j=1}^{2}. \tag{1.80}$$

For $\boldsymbol{w} \in D$ and $\boldsymbol{n} = (n_1, n_2)^T \in \mathbb{R}^2$ we put

$$\mathcal{P}(\boldsymbol{w}, \boldsymbol{n}) = \sum_{s=1}^{2} \boldsymbol{f}_s(\boldsymbol{w}) n_s, \tag{1.81}$$

which is the flux of the quantity $\boldsymbol{w}$ in the direction $\boldsymbol{n}$. The Jacobi matrix $D\mathcal{P}(\boldsymbol{w}, \boldsymbol{n}) / D\boldsymbol{w}$ can be expressed in the form

$$\frac{D\mathcal{P}(\boldsymbol{w}, \boldsymbol{n})}{D\boldsymbol{w}} = \mathbb{P}(\boldsymbol{w}, \boldsymbol{n}) = \sum_{s=1}^{2} \mathbb{A}_s(\boldsymbol{w}) n_s. \tag{1.82}$$

Lemma 1.5
The vector-valued functions $\boldsymbol{f}_s, s = 1, 2$, defined by (1.75) - (1.76), are first order homogenous mappings:

$$\boldsymbol{f}_s(\alpha \boldsymbol{w}) = \alpha \boldsymbol{f}_s(\boldsymbol{w}),\ \alpha > 0. \tag{1.83}$$

Moreover, we have

$$\boldsymbol{f}_s(\boldsymbol{w}) = \mathbb{A}_s(\boldsymbol{w}) \boldsymbol{w}. \tag{1.84}$$

Similarly,

$$\mathcal{P}(\alpha \boldsymbol{w}, \boldsymbol{n}) = \alpha \mathcal{P}(\boldsymbol{w}, \boldsymbol{n}),\ \alpha > 0, \tag{1.85}$$

$$\mathcal{P}(\boldsymbol{w}, \boldsymbol{n}) = \mathbb{P}(\boldsymbol{w}, \boldsymbol{n}) \boldsymbol{w}. \tag{1.86}$$

Proof: Relation (1.83) immediately follows from (1.75) - (1.76). Since $\boldsymbol{f}_s \in \mathcal{C}^1(D)^m$, the expression $(D\boldsymbol{f}_s(\boldsymbol{w})/D\boldsymbol{w})\,\boldsymbol{w} = \mathbb{A}_s(\boldsymbol{w})\,\boldsymbol{w}$ is the derivative of $\boldsymbol{f}_s$ in the direction $\boldsymbol{w}$ at the point $\boldsymbol{w}$. By the definition of the derivative and (1.83),

$$\mathbb{A}_s(\boldsymbol{w})\,\boldsymbol{w} = \lim_{\alpha\to 0}\frac{\boldsymbol{f}_s(\boldsymbol{w}+\alpha\boldsymbol{w})-\boldsymbol{f}_s(\boldsymbol{w})}{\alpha} \tag{1.87}$$

$$= \lim_{\alpha\to 0}\frac{(1+\alpha)\,\boldsymbol{f}_s(\boldsymbol{w})-\boldsymbol{f}_s(\boldsymbol{w})}{\alpha} = \boldsymbol{f}_s(\boldsymbol{w}). \tag{1.88}$$

The rest of the Lemma follows from the definitions of $\mathcal{P}$ and $\mathbb{P}$ and the above results. The flux Jacobians have the following form:

$$\mathbb{A}_1(\boldsymbol{w}) = \frac{D\boldsymbol{f}_1}{D\boldsymbol{w}}$$

$$= \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{w_2^2}{w_1^2}+\frac{R}{c_v}\frac{w_2^2+w_3^2}{2w_1^2} & \frac{2w_2}{w_1}-\frac{R}{c_v}\frac{w_2}{w_1} & -\frac{R}{c_v}\frac{w_3}{w_1} & \frac{R}{c_v} \\ -\frac{w_2w_3}{w_1^2} & \frac{w_3}{w_1} & \frac{w_2}{w_1} & 0 \\ -\frac{w_2w_4}{w_1^2}-\frac{w_2}{w_1^2}\frac{R}{c_v}\left(w_4-\frac{w_2^2+w_3^2}{2w_1}\right)+\frac{w_2}{w_1}\frac{R}{c_v}\frac{w_2^2+w_3^2}{2w_1^2} & \frac{w_4}{w_1}+\frac{1}{w_1}\frac{R}{c_v}\left(w_4-\frac{w_2^2+w_3^2}{2w_1}\right)-\frac{R}{c_v}\frac{w_2^2}{w_1^2} & -\frac{R}{c_v}\frac{w_2w_3}{w_1^2} & \frac{w_2}{w_1}+\frac{R}{c_v}\frac{w_2}{w_1} \end{pmatrix},$$

$$\mathbb{A}_2(\boldsymbol{w}) = \frac{D\boldsymbol{f}_2}{D\boldsymbol{w}}$$

$$= \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\frac{w_3w_2}{w_1^2} & \frac{w_3}{w_1} & \frac{w_2}{w_1} & 0 \\ -\frac{w_3^2}{w_1^2}+\frac{R}{c_v}\frac{w_2^2+w_3^2}{2w_1^2} & -\frac{R}{c_v}\frac{w_2}{w_1} & \frac{2w_3}{w_1}-\frac{R}{c_v}\frac{w_3}{w_1} & \frac{R}{c_v} \\ -\frac{w_3w_4}{w_1^2}-\frac{w_3}{w_1^2}\frac{R}{c_v}\left(w_4-\frac{w_2^2+w_3^2}{2w_1}\right)+\frac{w_3}{w_1}\frac{R}{c_v}\frac{w_2^2+w_3^2}{2w_1^2} & -\frac{R}{c_v}\frac{w_3w_2}{w_1^2} & \frac{w_4}{w_1}+\frac{1}{w_1}\frac{R}{c_v}\left(w_4-\frac{w_2^2+w_3^2}{2w_1}\right)-\frac{R}{c_v}\frac{w_3^2}{w_1^2} & \frac{w_3}{w_1}+\frac{R}{c_v}\frac{w_3}{w_1} \end{pmatrix}.$$

1.2.3 Rotational invariance

The rotational invariance of the Euler equations is represented by the relations

$$\mathcal{P}(\boldsymbol{w},\boldsymbol{n}) = \sum_{s=1}^{2}\boldsymbol{f}_s(\boldsymbol{w})\,n_s = \mathbb{Q}^{-1}(\boldsymbol{n})\,\boldsymbol{f}_1(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w}), \tag{1.89}$$

$$\mathbb{P}(\boldsymbol{w},\boldsymbol{n}) = \sum_{s=1}^{2}\mathbb{A}_s(\boldsymbol{w})\,n_s = \mathbb{Q}^{-1}(\boldsymbol{n})\,\mathbb{A}_1(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w})\,\mathbb{Q}(\boldsymbol{n}), \tag{1.90}$$

$$\boldsymbol{n} = (n_1,n_2)\in\mathbb{R}^2, |\boldsymbol{n}|=1, \boldsymbol{w}\in D, \tag{1.91}$$

where

$$\mathbb{Q}(\boldsymbol{n}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & n_1 & n_2 & 0 \\ 0 & -n_2 & n_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.92}$$

1.2.4 Diagonalization of the Jacobi matrix

We have

$$\boldsymbol{f}_1(\boldsymbol{w}) = \left(w_1, w_2^2/w_1 + (\gamma-1)\left[w_4 - \left(w_2^2+w_3^2\right)/(2w_1)\right],\right. \tag{1.93}$$
$$\left. w_2w_3/w_1, w_2\left[\gamma w_4 - (\gamma-1)\left(w_2^2+w_3^2\right)/(2w_1)\right]/w_2\right)^T,$$

and, with the notation $\boldsymbol{v} = (u, v)$,

$$\mathbb{A}_1(\boldsymbol{w}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{\gamma-1}{2}|\boldsymbol{v}|^2 - u^2 & (3-\gamma)u & (1-\gamma)v & 1-\gamma \\ -uv & v & u & 0 \\ u\left((\gamma-1)|\boldsymbol{v}|^2 - \gamma\frac{E}{\rho}\right) & \gamma\frac{E}{\rho} - (\gamma-1)u^2 - \frac{\gamma-1}{2}|\boldsymbol{v}|^2 & (1-\gamma)uv & \gamma u \end{pmatrix}. \tag{1.94}$$

The matrix $\mathbb{A}_1(\boldsymbol{w})$ has eigenvalues

$$\tilde{\lambda}_1(\boldsymbol{w}) = u - a, \tilde{\lambda}_2(\boldsymbol{w}) = \tilde{\lambda}_3(\boldsymbol{w}) = u, \tilde{\lambda}_4(\boldsymbol{w}) = u + a \tag{1.95}$$

and the corresponding eigenvectors are

$$\boldsymbol{r}_1(\boldsymbol{w}) = \left(1, u-a, v, |\boldsymbol{v}|^2/2 + a^2(\gamma-1) - ua\right)^T, \tag{1.96}$$
$$\boldsymbol{r}_2(\boldsymbol{w}) = \left(1, u, v, |\boldsymbol{v}|^2/2\right)^T, \tag{1.97}$$
$$\boldsymbol{r}_3(\boldsymbol{w}) = \left(1, u, v-a, |\boldsymbol{v}|^2/2 - va\right)^T, \tag{1.98}$$
$$\boldsymbol{r}_4(\boldsymbol{w}) = \left(1, u+a, v, |\boldsymbol{v}|^2/2 + a^2(\gamma-1) + ua\right)^T. \tag{1.99}$$

These formulas are possible to be verified. As the eigenvectors $\boldsymbol{r}_s(\boldsymbol{w}), s = 1, ..., 4$, are linearly independent for each $\boldsymbol{w} \in D$, we can write

$$\tilde{\mathbb{T}}^{-1}(\boldsymbol{w})\,\mathbb{A}_1(\boldsymbol{w})\,\tilde{\mathbb{T}}(\boldsymbol{w}) = \tilde{\Lambda}(\boldsymbol{w}), \tag{1.100}$$

where the matrix $\tilde{\mathbb{T}}(\boldsymbol{w})$ has the vectors $\boldsymbol{r}_s(\boldsymbol{w}), s = 1, ..., 4$, as its columns and

$$\tilde{\Lambda}(\boldsymbol{w}) = \operatorname{diag}\left(\tilde{\lambda}_1(\boldsymbol{w}), ..., \tilde{\lambda}_4(\boldsymbol{w})\right). \tag{1.101}$$

This means that the matrix $\mathbb{A}_1(\boldsymbol{w})$ is diagonalizable.

Now we can show that the matrix $\mathbb{P}(\boldsymbol{w}, \boldsymbol{n})$ is diagonalizable too. By (1.90) and (1.100)

$$\mathbb{P}(\boldsymbol{w}, \boldsymbol{n}) = |\boldsymbol{n}|\,\mathbb{Q}^{-1}(\boldsymbol{n})\,\tilde{\mathbb{T}}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w})\,\tilde{\Lambda}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w})\,\tilde{\mathbb{T}}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w})^{-1}\,\mathbb{Q}(\boldsymbol{n}). \tag{1.102}$$

Under the notation

$$\Lambda(\boldsymbol{w}, \boldsymbol{n}) = |\boldsymbol{n}|\,\tilde{\Lambda}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w}) = \operatorname{diag}|\boldsymbol{n}|\left(\tilde{\Lambda_1}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w}), ..., \tilde{\Lambda_4}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w})\right),$$
$$\mathbb{T}(\boldsymbol{w}, \boldsymbol{n}) = \mathbb{Q}^{-1}(\boldsymbol{n})\,\tilde{\mathbb{T}}(\mathbb{Q}(\boldsymbol{n})\,\boldsymbol{w}), \tag{1.103}$$

we see that the matrix $\mathbb{P}(\boldsymbol{w}, \boldsymbol{n})$ satisfies the relation $\mathbb{T}^{-1}\mathbb{P}\mathbb{T} = \Lambda(\boldsymbol{w}, \boldsymbol{n}) = \operatorname{diag}(\lambda_1, ..., \lambda_4)$ and thus $\mathbb{T}$ diagonalizes the matrix $\mathbb{P}$.

Chapter 2

Computational methods in fluid dynamics

In this section we shall consider an initial-boundary value problem for the system of equations in the space-time cylinder $Q_T = \Omega \times (0, T)$, where $\Omega \in \mathbb{R}^2$. We give here a short overview of methods for discretization in space.

2.1 Triangulation

First step in the process of the discretization is to divide the closure $\overline{\Omega}$ of the computational domain Ω into a finite number of subsets with properties described below. These subsets form the set, further denoted by $\mathcal{T}_h$, called the *triangulation of the domain* Ω. The parameter $h > 0$ of the triangulation usually represents maximum of diameters of all elements $K \in \mathcal{T}_h$. The elements $K \in \mathcal{T}_h$ are in the context of the finite volume method called *finite volumes*.

Properties of $\mathcal{T}_h$:

a) Each $K \in \mathcal{T}_h$ is closed and connected with its interior $K^\circ \neq \emptyset$.

b) Each $K \in \mathcal{T}_h$ has a Lipschitz boundary.

c) $\cup_{K \in \mathcal{T}_h} K = \overline{\Omega}$

d) If $K_1, K_2 \in \mathcal{T}_h$, $K_1 \neq K_2$, then $K_1^\circ \cap T_2^\circ = \emptyset$.

In our case of the two-dimensional problem, we assume that the domain Ω is obtained as an approximation of the original computational domain (also denoted by Ω), and the triangulation is chosen accordingly to the following attributes:

A) Each $K \in \mathcal{T}_h$ is a closed triangle or quadrilateral, possibly with curved edges.

B) For $K_1, K_2 \in \mathcal{T}_h$, $K_1 \neq K_2$ we have either $K_1 \cap K_2 = \emptyset$ or K_1, K_2 share one vertex or K_1, K_2 share one edge. If every edge of the mesh that does not lie on the boundary $\partial\Omega$ is a common edge of two elements K_i and K_j, then we call the mesh *regular*, otherwise we call the mesh *irregular*.

C) $\cup_{K\in\mathcal{T}_h} K = \overline{\Omega}$.

Furthermore

$$\mathcal{T}_h = \{K_i, i \in I\},$$

where $I \subset Z^+ = \{0, 1, 2, ...\}$ is a suitable index set.
By Γ_{ij} we denote a common edge between two neighboring elements K_i and K_j. We set

$$s(i) = \{j \in I; K_j \text{ is a neighbor of } K_i\}.$$

The boundary $\partial\Omega$ is formed by a finite number of edges of elements K_i adjacent to $\partial\Omega$. We denote all these boundary edges by S_j, where $j \in I_b \subset Z^- = \{-1, -2, ...\}$. Now we set

$$\gamma(i) = \{j \in I_b; S_j \text{ is a edge of } K_i \in \mathcal{T}_h\}$$

and

$$\Gamma_{ij} = S_j \text{ for } K_i \in \mathcal{T}_h \text{ such that } S_j \subset \partial K_i, j \in I_b.$$

For K_i not containing any boundary edge S_j we set $\gamma(i) = \emptyset$.
Obviously, $s(i) \cap \gamma(i) = \emptyset$ for all $i \in I$. If we write $S(i) = s(i) \cup \gamma(i)$, we have

$$\partial K_i = \cup_{j\in S(i)} \Gamma_{ij}, \quad \partial K_i \cap \partial\Omega = \cup_{j\in\gamma(i)} \Gamma_{ij}.$$

In what follows, by $\boldsymbol{n}_{ij}(x_1, x_2)$ we shall denote the unit outer normal to K_i on Γ_{ij} at the point (x_1, x_2).

Model example Our model example in this chapter will be the scalar *linear advection-diffusion equation* with diffusion term left out:

$$\nabla \cdot \boldsymbol{f}(u(x_1, x_2)) = 0. \tag{2.1}$$

In particular, we will be concerned with the case

$$\boldsymbol{f}(u) = \beta u,$$

where $\beta(x_1, x_2) = (-x_2, x_1)/|\boldsymbol{x}|$ represents a circular counterclockwise flow field and $\Omega = [0, 1] \times [0, 1]$. We prescribe the following boundary condition:

$$u = g \text{ on } \Gamma_-, \text{ where} \tag{2.2}$$
$$g = 1 \text{ on } \Gamma_-^1, \text{ and} \tag{2.3}$$
$$g = 0 \text{ on } \Gamma_-^2, \tag{2.4}$$

where $\Gamma_- = \{\boldsymbol{x} = (x_1, x_2) \in \partial\Omega : \beta(x_1, x_2) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\}$, $\Gamma_1 = [0, 0.5] \times 0$, and $\Gamma_-^2 = \Gamma_- \setminus \Gamma_-^1$. We do not prescribe any boundary condition on the rest of the boundary, i.e. on $\Gamma = \partial\Omega \setminus \Gamma_-$.

2.2 Finite Volume method

2.2.1 Scheme derivation

Let us assume that $u : \overline{\Omega} \to \mathbb{R}$ is a classical $\mathcal{C}^1$ solution of the equation (2.1), $\mathcal{T}_h = \{K_i\}_{i \in I}$ is a mesh with the above properties. Integrating the equation (2.1) over any K_i and using Green's theorem on K_i, we get the identity

$$\int_{\partial K_i} \boldsymbol{f}(u) \cdot \boldsymbol{n} dS = 0. \tag{2.5}$$

Now we shall approximate the integral averages $\int_{K_i} u(\boldsymbol{x})\, d\boldsymbol{x} \setminus |K_i|$ of the quantity u over the finite volume K_i by u_i:

$$u_i \approx \frac{1}{|K_i|} \int_{K_i} u(\boldsymbol{x})\, d\boldsymbol{x}, \tag{2.6}$$

called the value of *the approximate solution* on K_i. Futher, we approximate the flux $\boldsymbol{f}(u) \cdot (\boldsymbol{n}_{ij})$ of the quantity u through the edge Γ_{ij} in the direction $\boldsymbol{n}_{ij}$ with the aid of a *numerical flux* $\boldsymbol{H}(u_i, u_j, \boldsymbol{n}_{ij})$, depending on the value of the approximate solution u_i on the finite volume K_i, the value u_j on K_j, and on the normal $\boldsymbol{n}_{ij}$. If $\Gamma_{ij} \subset \partial\Omega$ (i.e. the finite volume K_i is adjacent to $\partial\Omega$, $j \in \gamma(i)$), then there is no neighbor K_j of K_i adjacent to the edge Γ_{ij} from the exterior of Γ, and it is necessary to specify u_j on the basis of boundary conditions. This will be explained further in the section about the discretization of the Euler equations.

2.2.2 Properties of the numerical flux

In what follows, we shall assume that the numerical flux $\boldsymbol{H}$ has the following properties:

A) $\boldsymbol{H}(u, v, \boldsymbol{n})$ is defined and continuous on $\mathcal{D} \times \mathcal{D} \times \mathcal{S}_1$, where $\mathcal{D}$ is the domain of definition of the flux $\boldsymbol{f}$ and $\mathcal{S}_1$ is the unit sphere in $\mathbb{R}^N$ (here $N = 2$).

B) $\boldsymbol{H}$ is *consistent*:

$$\boldsymbol{H}(u, u, \boldsymbol{n}) = \boldsymbol{f}(u) \cdot \boldsymbol{n},\ u \in \mathcal{D},\ \boldsymbol{n} \in \mathcal{S}_1.$$

C) $\boldsymbol{H}$ is *conservative*:

$$\boldsymbol{H}(u, v, \boldsymbol{n}) = -\boldsymbol{H}(v, u, -\boldsymbol{n}),\ u, v \in \mathcal{D},\ \boldsymbol{n} \in \mathcal{S}_1.$$

We define a finite volume approximate solution of (2.1) *as piecewise constant function* u_h, *defined a.e. in* Ω *so that* $u_h|_{K_i{}^o} = u_i$ *for all* $i \in I$, *where* $K_i{}^o$ *is the interior*

of K_i, i.e. $K_i{}^o = K_i \backslash \partial K_i$, and u_i are obtained from the formula

$$\frac{1}{K_i} \sum_{j \in S(i)} \boldsymbol{H}(u_i, u_j, \boldsymbol{n}_{ij}) = 0, \quad K_i \in \mathcal{T}_h. \tag{2.7}$$

The number u_i is the value of the approximate solution on the finite volume K_i.

2.2.3 Example

Now we give an example of a numerical flux, which is the *upwinding* numerical flux defined by

$$H\left(u_i, u_j, n_{ij}\right) = \begin{cases} u_i, & \text{if } \beta \cdot \boldsymbol{n}_{ij} > 0 \\ u_j, & \text{if } \beta \cdot \boldsymbol{n}_{ij} \leq 0 \end{cases}. \tag{2.8}$$

Solutions of the model example using the finite volume method with the numerical flux (2.8) are presented below. In this case the equations (2.7) are linear. The system was solved using the direct solver from the UMFPACK package. The computation was obtained on three different meshes.

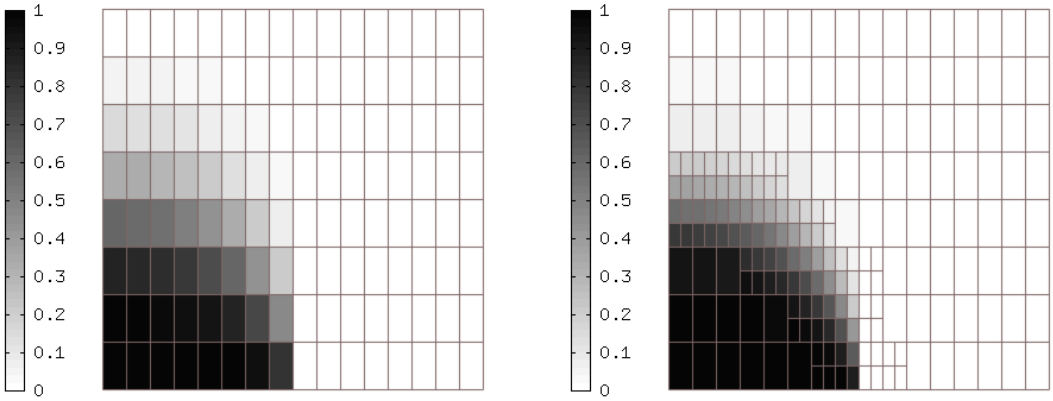

Figure 2.1: Solution u on a mesh with 128 (left), and 206 (right) elements.

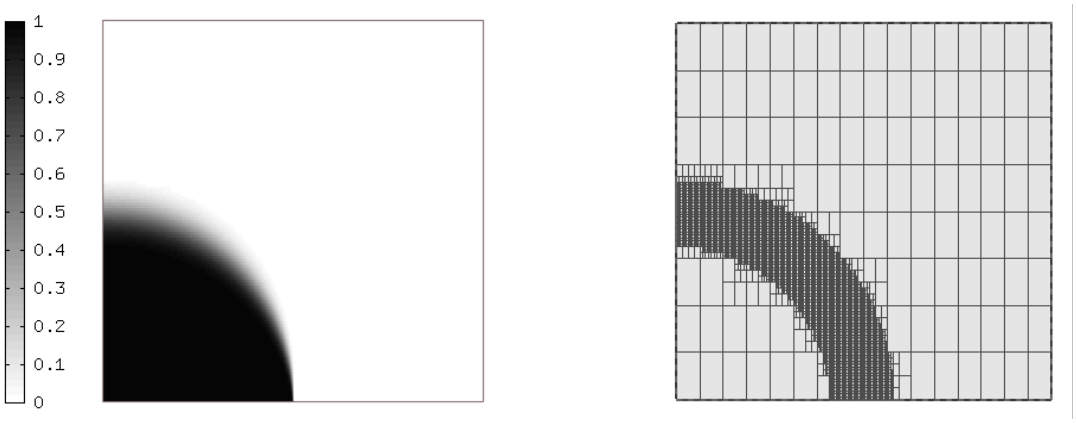

Figure 2.2: Solution u (left) and the corresponding mesh (right) with 4751 elements.

We can see that the solution is smeared near the steep gradient, even if we used a greatly refined mesh. We would like to avoid this behavior, and get better resolution of the steep gradient.

2.3 Finite Element Method

Application of the standard FEM to the problem (2.1) is very straightforward. We multiply the equation (2.1) with a test function v from the following finite-dimensional approximation of the $H^1(\Omega)$ space:

$$V = \left\{ \varphi \in H^1(\Omega); \varphi = \sum_{0 \leq i,j \leq p} \alpha_{ij}\ x_1^i\ x_2^j,\ \alpha_{ij} \in \mathbb{R} \right\}, \tag{2.9}$$

where $p \in \mathbb{N}$. We then integrate over Ω and use the Green's theorem. We get the following identity:

$$\int_\Omega \boldsymbol{f}(u) \cdot \nabla v d\boldsymbol{x} = \int_{\Gamma_-} \beta(g)\, v \cdot \boldsymbol{n}\ \forall v \in V. \tag{2.10}$$

For singularly perturbed equations, or strictly first order equations, such as our model equation, the standard Galerkin FEM exhibits *Gibbs phenomenon*, manifested by *spurious oscillations* in the numerical solution. This behavior is well noticeable in the following figures.

2.3.1 Example

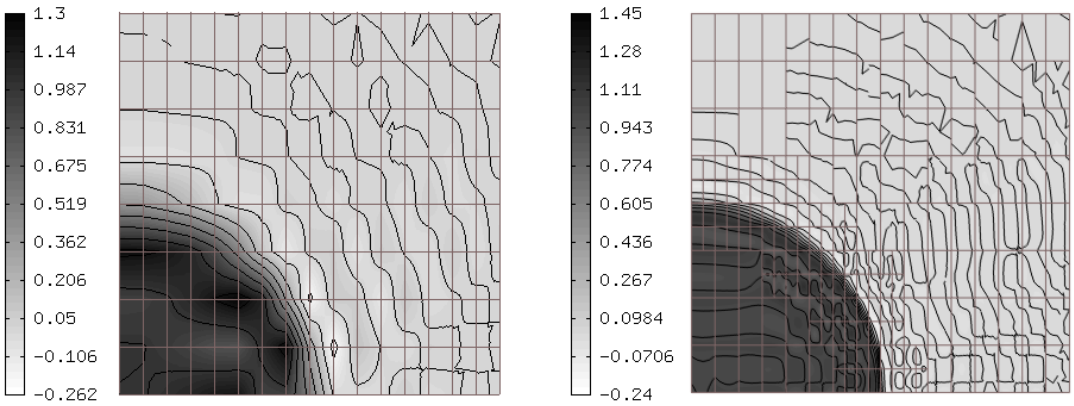

Figure 2.3: Solution u on a mesh with 128 elements and p = 1 (left), and 206 elements and p = 2 (right).

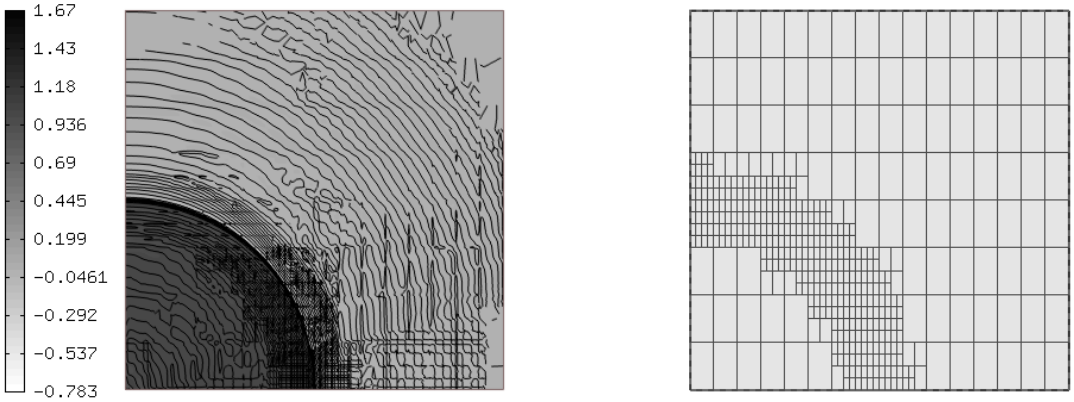

Figure 2.4: Solution u (left) and the corresponding mesh (right) with 449 elements, p = 5, and 11206 degrees of freedom.

The oscillations can be diminished using a stabilization technique, e.g. streamwind-upwind Petrov-Galerking stabilization (SUPG). Their disadvantage is the necessity to tune various parameters in order that the resulting schemes function properly. For a reference, see e.g. [12], [15].

2.4 Discontinuous Galerkin method

For complex problems of compressible flow, the choice of optimal stabilization parameters becomes quite sophisticated and difficult. Due to this reason, there was an

effort to develop methods which would not need such stabilization techniques, and would still offer reasonable resolution of shockwaves, boundary and interior layers, and steep gradients without exhibiting spurious oscillations in an approximate solution. It is based on the idea to combine finite volume and finite element methods leading to the so-called *discontinuous Galerkin finite element method (DGFEM)*. Here we shall derive and analyze DGFEM for our model example. Let $\mathcal{T}_h$ be a triangulation of Ω. For each $K \in \mathcal{T}_h$ we introduce the notation

$$\partial K^- = \{x \in \partial K; \beta(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\}, \tag{2.11}$$

$$\partial K^+ = \{x \in \partial K; \beta(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) \geq 0\}. \tag{2.12}$$

By $H^k(\Omega, \mathcal{T}_h)$ we denote the so-called *broken Sobolev space*:

$$H^k(\Omega, \mathcal{T}_h) = \left\{v \in L^2(\Omega);\ v|_K \in H^k(K)\, \forall K \in \mathcal{T}_h\right\}. \tag{2.13}$$

For $u \in H^1(\Omega, \mathcal{T}_h)$ we set

$$u_K^+ = \text{trace of } u|_K \text{ on } \partial K \tag{2.14}$$

(i.e. the interior trace of u on ∂K). For each edge $E \subset \partial K \backslash \Gamma$ of K, there exists $K' \neq K,\ K' \in \mathcal{T}_h$, adjacent to E from the opposite side than K. Then we put

$$u_K^- = \text{trace of } u|_{K'} \text{ on } E. \tag{2.15}$$

In this way we obtain the exterior trace u_K^- of u on $\partial K \backslash \Gamma$ and define the jump of u on $\partial K \backslash \Gamma$:

$$[u]_K = u_K^+ - u_K^-. \tag{2.16}$$

2.4.1 Derivation of the DGFEM

Let $u \in H^1(\Omega)$ be a solution of the problem (2.1). Then u satisfies the identity

$$\int_K \nabla \cdot (\beta u)\, \varphi d\boldsymbol{x} = 0, \quad \varphi \in H^1(\Omega, \mathcal{T}_h),\ K \in \mathcal{T}_h. \tag{2.17}$$

The application of Green's theorem gives

$$\begin{aligned} \int_K \nabla \cdot (\beta u)\, \varphi d\boldsymbol{x} &= \int_{\partial K} \left(\beta u_K^+\right) \cdot \boldsymbol{n} \varphi_K^+ dS - \int_K (\beta u) \cdot \nabla \varphi d\boldsymbol{x} \\ &= \int_{\partial K^-} \left(\beta u_K^+\right) \cdot \boldsymbol{n} \varphi_K^+ dS + \int_{\partial K^+} \left(\beta u_K^+\right) \cdot \boldsymbol{n} \varphi_K^+ dS \\ &- \int_K (\beta u) \cdot \nabla \varphi d\boldsymbol{x}, \end{aligned} \tag{2.18}$$

where $\boldsymbol{n}$ is the unit outer normal to the element boundary ∂K. As $u \in H^1(\Omega)$, we have $u_K^- = u_K^+$. Moreover, $u_K^-|_{\partial K^- \cap \Gamma} := u|_{\partial K^- \cap \Gamma} = g$. Then we can write

$$\begin{aligned} \int_K \nabla \cdot (\beta u)\, \varphi d\boldsymbol{x} &= \int_{\partial K^-} \left(\beta u_K^-\right) \cdot \boldsymbol{n} \varphi_K^+ dS + \int_{\partial K^+} \left(\beta u_K^+\right) \cdot \boldsymbol{n} \varphi_K^+ dS \\ &- \int_K (\beta u) \cdot \nabla \varphi d\boldsymbol{x}. \end{aligned} \tag{2.19}$$

Applying Green's theorem again, we get the identity

$$\int_K \nabla \cdot (\beta u)\, \varphi d\boldsymbol{x} = \int_{\partial K^-} \left(\beta \left(u_K^+ - u_K^-\right)\right) \cdot \boldsymbol{n} \varphi_K^+ dS. \tag{2.20}$$

Setting

$$\begin{aligned} a_K(u, \varphi) &= \int_K \nabla \cdot (\beta u)\, \varphi d\boldsymbol{x} - \int_{\partial K^- \setminus \Gamma} (\beta [u]_K) \cdot \boldsymbol{n} \varphi_K^+ dS \\ &- \int_{\partial K^- \cap \Gamma_-} \left(\beta u_K^+\right) \cdot \boldsymbol{n} \varphi_K^+ dS, \end{aligned} \tag{2.21}$$

$$L_K(\varphi) = \int_{\partial K^- \cap \Gamma_-} (\beta g) \cdot \boldsymbol{n} \varphi_K^+ dS, \tag{2.22}$$

we can rewrite the equation (2.20) as

$$a_K(u, \varphi) = L_K(\varphi), \quad \varphi \in H^1(K), \quad K \in \mathcal{T}_h. \tag{2.23}$$

This identity makes sense also for $u \in H^1(\Omega, \mathcal{T}_h)$. In this case, we can note that in (2.18), on ∂K^- (= the inlet of K with respect to the velocity β) we replace the value u_K^+ (the interior trace of u) by u_K^-. This means that *upwinding* is used here, because the value of the trace of u on ∂K^- is taken from the side of ∂K^- against the velocity direction.

Now, on the basis of (2.23) we come to the definition of the *discrete problem*. The *approximate solution* is a function u_h satisfying the conditions

A) $u_h \in V_h = V^{p,-1}(\Omega, \mathcal{T}_h) := \{\varphi \in L^2(\Omega); \varphi|_K \in P^p(K)\, \forall K \in \mathcal{T}_h\}$

B) $a_K(u_h, \varphi_h) = L_K(\varphi_h), \ \forall \varphi_h \in V_h, \ \forall K \in \mathcal{T}_h.$

Here, p is an integer. In general, on each element a different polynomial degree can be used for the approximation. The approximate solution and test functions are piecewise polynomial functions without any continuity requirement on interfaces between neighboring elements. The continuity requirement is replaced here by the jump term $\int_{\partial K^- \setminus \Gamma} (\beta [u]_K) \cdot \boldsymbol{n} \varphi_K^+ dS$.

In Figures 2.5, 2.6 we can see that we successfully avoided the smeared layer close to the steep gradient, and also the Gibbs phenomenon in the whole domain. What we have to deal here with on the other hand are overshoots and undershoots close to the step gradient. There is several approaches, some of which are described in the last chapter with numerical experiments, for the case of the Euler equations.

2.4.2 Example

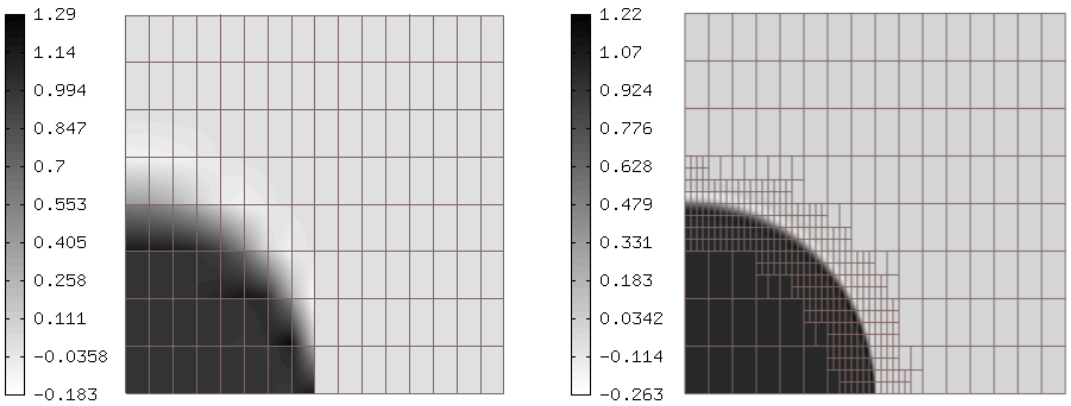

Figure 2.5: Solution u on a mesh with 128 elements and p = 1 (left), and 206 elements and p = 2 (right).

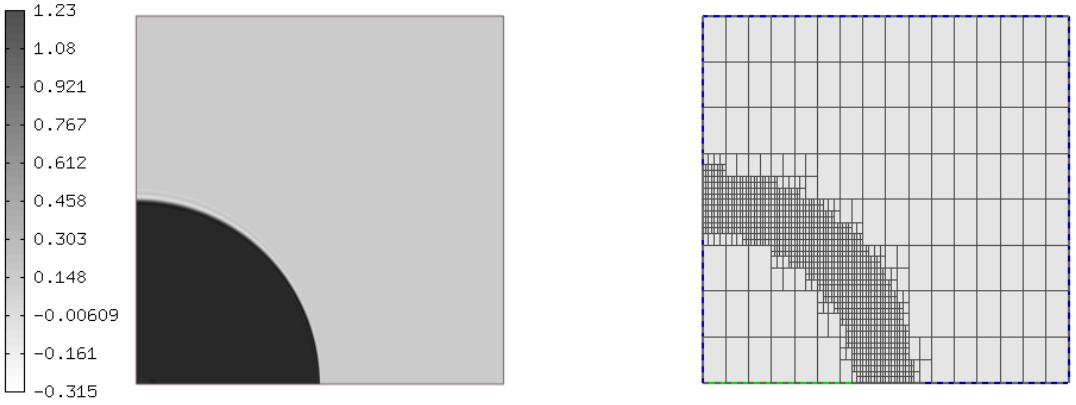

Figure 2.6: Solution u (left) and the corresponding mesh (right) with 1271 elements, p = 3, and 20336 degrees of freedom.

Chapter 3

Adaptivity algorithms

In computational fluid dynamics we are interested in the computation of sufficiently accurate solutions of various flow problems. The goal of the design of any numerical method is *reliability* and *efficiency*. Reliability means that the computational error is controlled at a given tolerance level. The efficiency means that the cost of the computation of a solution within a given tolerance is as small as possible. These two requirements are usually achieved with the aid of mesh refinement techniques. The goal is to achieve reliability either in the sense that the numerical solution approximates the exact solution in a given norm within a given tolerance, or in the sense that some physically relevant quantities (e.g. flux through a part of a boundary, drag, lift) are computed within a given tolerance.

Standard h-adaptive FEM, where (adaptive) mesh refinement is used, is a well known, and well spread method. Sometimes, much faster convergence can be achieved by increasing the polynomial degree of the elements instead (p-refinement). Such approach is far more efficient for elements of the mesh where the solution is smooth. In the following section, a brief description of this technique is presented.

3.1 p-adaptivity

In the process of FEM, or DGFEM usage, the domain Ω is first approximated by a suitable (often polygonal) domain Ω_h. This is one of the so-called *variational crimes* – departures from the "mathematically clean" variational framework – since $\Omega_h \not\subset \Omega$ and the solution or other functions from the weak formulation are not defined where they are to be approximated or evaluated. In practice these crimes are often hard to avoid. In what follows, we will denote the approximation Ω_h of the domain Ω used in computations also by Ω.

3.1.1 Higher-order finite element space

Let the domain Ω be covered with a mesh $\mathcal{T}_h = \{K_1, K_2, \ldots, K_M\}$ where the elements K_m carry arbitrary polynomial degrees $1 \leq p_m$, $m = 1, 2, \ldots, M$. The broken Sobolev space $H^1(\Omega, \mathcal{T}_h)$ is now approximated by a space of piecewise-polynomial functions

$$V_h = \{v \in L^2(\Omega);\ v|_{K_m} \in P^{p_m}(K_m) \text{ for all } 1 \leq m \leq M\}$$

where P^p is defined as

$$P^p = \text{span}\{\sum_{0 \leq i,j \leq p} \alpha_{ij}\ x_1^i\ x_2^j,\quad \alpha_{ij} \in \mathbb{R}\}$$

in case of quadrilaterals and

$$P^p = \text{span}\{\sum_{\substack{0 \leq i,j \leq p \\ i+j \leq p}} \alpha_{ij}\ x_1^i\ x_2^j,\quad \alpha_{ij} \in \mathbb{R}\}$$

in case of triangles.

Basis functions are defined with help of functions on *reference elements* that we call *shape functions*. Reference elements and their mapping onto physical elements is a standard approach described e.g. in [9]. We will denote the appropriate (i.e. triangular or quadrilateral) reference elements by $\hat{K}$. The reference elements are shown in figure 3.1

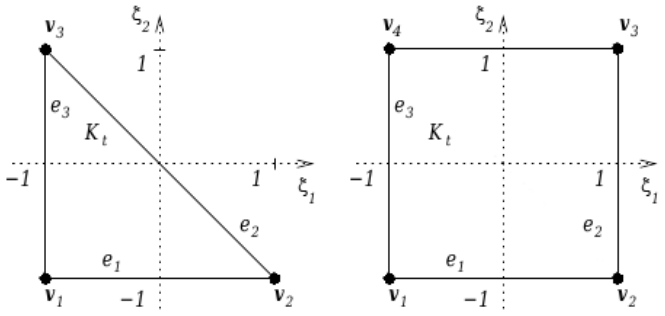

Figure 3.1: Reference triangle and reference quadrilateral.

The next section shows one possible way of constructing shape functions for the discontinuous Galerkin method.

3.1.2 Hierarchic shape functions

Hierarchic shape functions are constructed in such a way that the basis $\mathcal{B}^{p+1}$ of the polynomial space $P^{p+1}(\hat{K})$ is obtained from the basis $\mathcal{B}^p$ of the polynomial space $P^p(\hat{K})$ by adding new shape functions only. This is essential for p- and hp-adaptive finite element codes since one does not have to change the shape functions completely when increasing the degree of polynomial approximation. The bases formed by this approach shall be called *hierarchic bases*. In this section we will describe the popular *Legendre-based* hierarchic shape functions.

Definition 3.1 (Legendre polynomials) *We define the* Legendre polynomials *of degree k as*

$$\mathrm{L}_k(x) = \frac{1}{2^k k!}\frac{\mathrm{d}^k}{\mathrm{d}x^k}(x^2-1)^k, \quad k = 0, 1, 2, \ldots$$

The set of Legendre polynomials forms an orthonormal basis of the space $L^2(-1, 1)$.

Quadrilaterals On quadrilaterals, the set of shape functions is the product of Legendre polynomials. Displayed are shape functions from the bases $\mathcal{B}^p$ for $p = 1, 2, 3$.

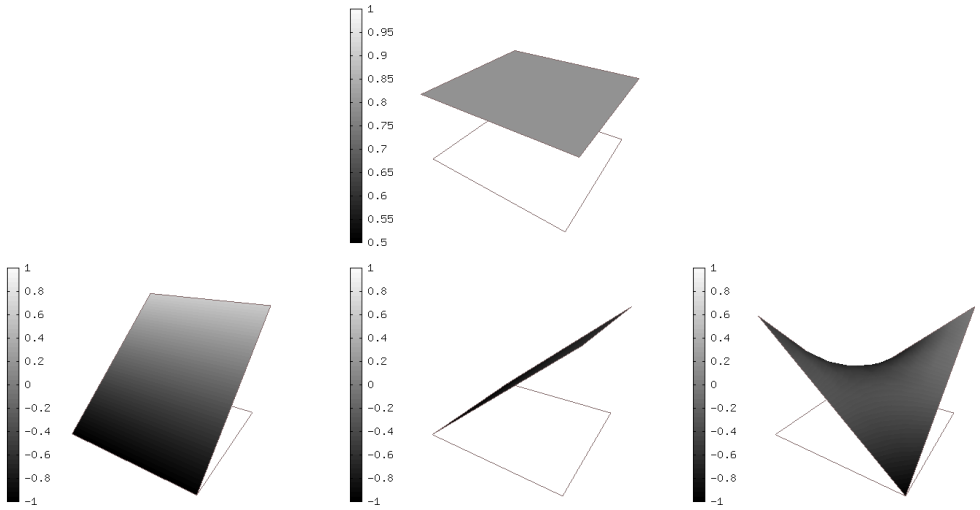

Figure 3.2: Basis $\mathcal{B}^0$ of the space P^0 (top), basis $\mathcal{B}^1$ of the space P^1 (bottom)

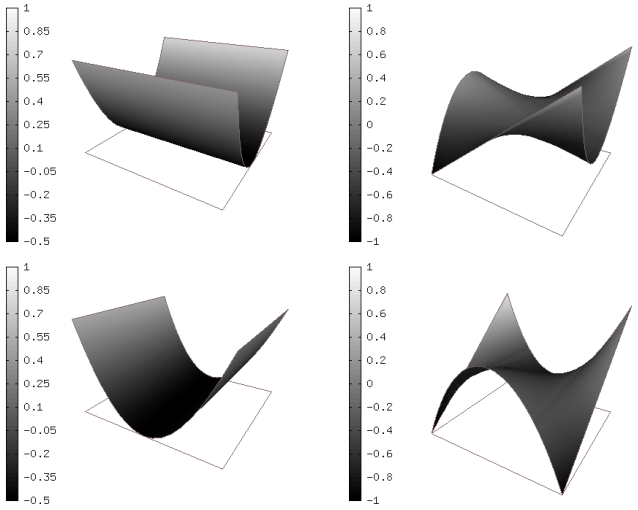

Figure 3.3: Basis $\mathcal{B}^2$ of the space P^2

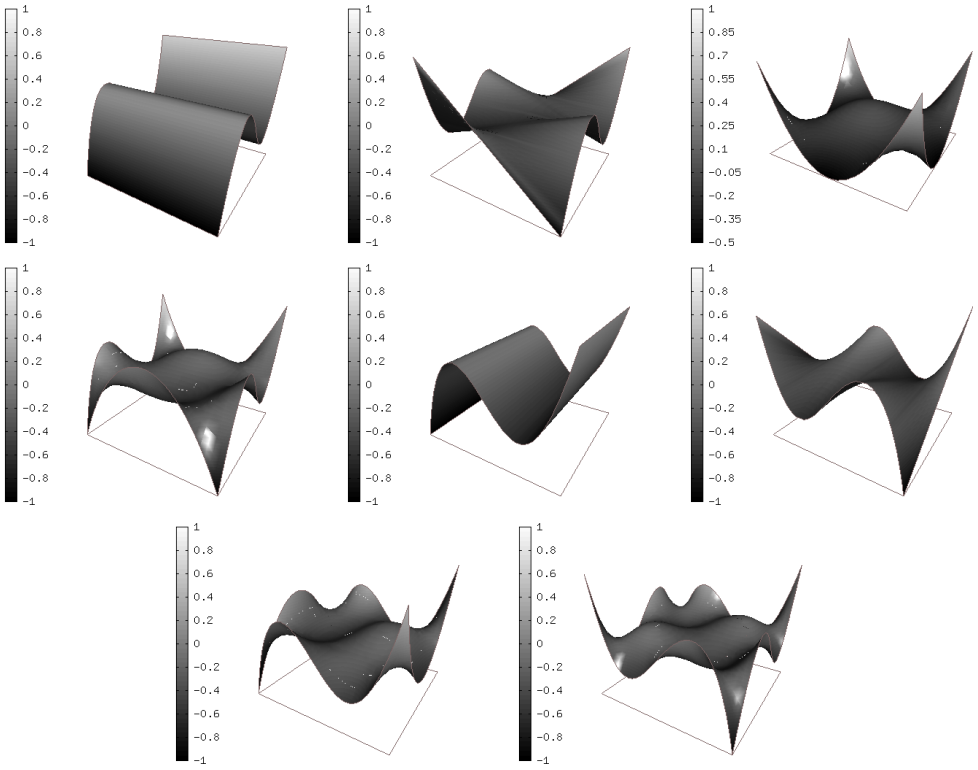

Figure 3.4: Basis $\mathcal{B}^3$ of the space P^3

Triangles Situation on triangles is more complicated, because the set of shape functions cannot be the product of Legendre polynomials. In the construction of the set, the following definition will be used.

Definition 3.2 (Affine coordinates) Barycentric coordinates *on the triangular reference domain* $\hat{K}$ *are defined as*

$$\lambda_1(\xi_1,\xi_2)=\frac{\xi_2+1}{2},\ \lambda_2(\xi_1,\xi_2)=-\frac{\xi_1+\xi_2}{2},\ \lambda_3(\xi_1,\xi_2)=\frac{\xi_1+1}{2}$$

Now the shape functions are defined in the following way:

$$L_{ij}^{tri}=\mathrm{L}_i(\lambda_3(x_1,x_2)-\lambda_2(x_1,x_2))\cdot\mathrm{L}_j(\lambda_2(x_1,x_2)-\lambda_1(x_1,x_2)), \tag{3.1}$$

the resulting set is then $L_p^{tri}=\left\{L_{ij}^{tri},\ i,j=0,...,p\right\}$, where p represents the highest polynomial degree we use for approximation. Displayed are shape functions from the bases $\mathcal{B}^p$ for $p=1,2,3$.

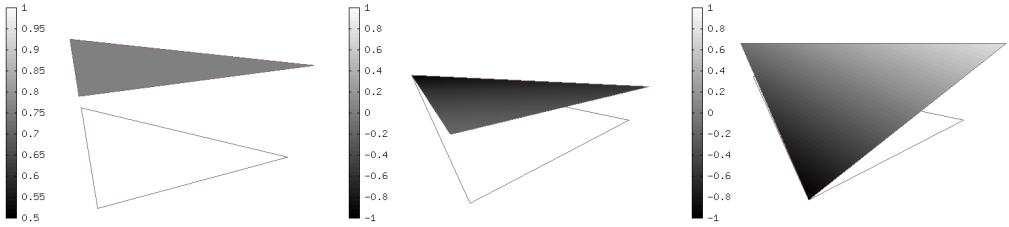

Figure 3.5: Basis $\mathcal{B}^0$ of the space P^0 (left), basis $\mathcal{B}^1$ of the space P^1 (rest)

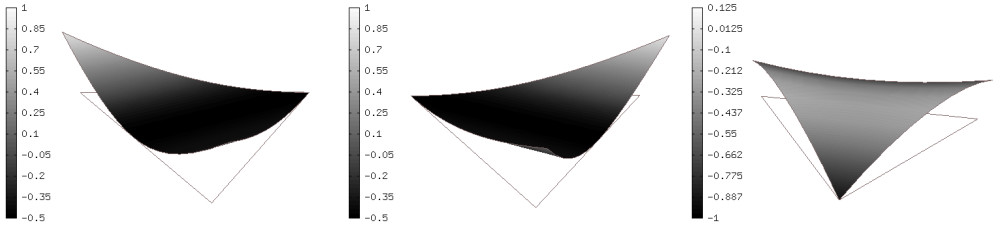

Figure 3.6: Basis $\mathcal{B}^2$ of the space P^2

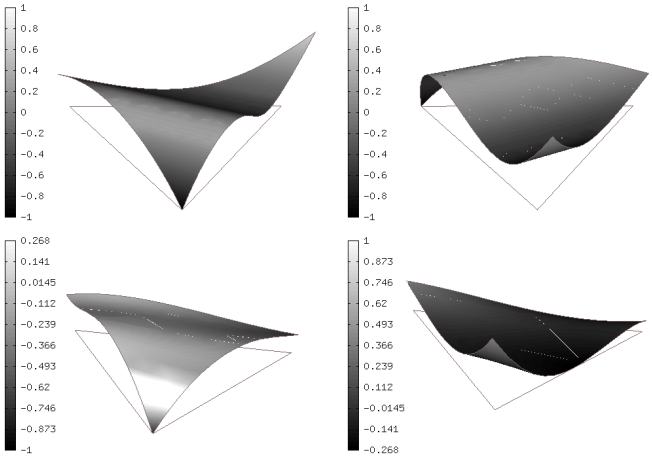

Figure 3.7: Basis $\mathcal{B}^3$ of the space P^3

Proposition 3.1 *The shape functions described above, both on quadrilaterals, and on triangles constitute a hierarchic basis of the space* $P^{p_m}(\hat{K})$. *Hierarchic in the sense that all functions from* $P^{p_m-1}(\hat{K})$ *belong to* $P^{p_m}(\hat{K})$.

Proof The complete proof can be found in [21]. Briefly, the following must be verified:

A) Are all the shape functions linearly independent?

B) Do they all belong to the space $P^{p_m}(\hat{K})$?

C) Matches their number the dimension of the space $P^{p_m}(\hat{K})$?

3.1.3 Higher-order numerical quadrature

Most commonly, the integrals that appear in our problems are evaluated numerically by the *Gaussian quadrature*. The k-point Gaussian quadrature rule on the domain $\hat{K}$ has the form

$$\int_{\hat{K}} g(\xi)\mathrm{d}\xi \approx \sum_{i=1}^{k} w_{k,i} g(\xi_{k,i}) \tag{3.2}$$

where g is a bounded continuous function, $\xi_{k,i} \in \hat{K}, i = 1, 2, \ldots, k$ are the integration points and $w_{k,i} \in \mathbb{R}$ are the integration weights. The sum of the integration weights must be equal to the area of $\hat{K}$, so that the rule (3.2) is exact for constants. If the points and weights are chosen carefully, the formula (3.2) can be exact for polynomials up to a certain degree $q > 0$.

In 1D the integration points are roots of the Legendre polynomials. Also in 2D it is possible to find the integration points and weights for low-degree polynomials ocurring in traditional linear FEM. For higher-degree polynomials, however, the task of finding optimal Gaussian quadrature rules presents a complex non-linear optimization problem with many open questions left. Optimal integration points and weights are known on $\hat{K}$ for polynomials up to degree 10. Suboptimal (with more points than necessary) rules have been found for polynomials up to degree 20. Complete integration tables along with more information on this subject can be found in [21].

3.2 Hanging nodes

As the goal is automatic mesh adaptivity on both triangular and quadrilateral (or combined) meshes, one of the important characteristics of triangulations is their regularity. This section shows why the regularity assumption (from 2.1) should be dropped and how irregular meshes are treated.

3.2.1 Hanging nodes and irregularity rules

At the beginning, let us recall the *red-green* refinement strategy. This technique first subdivides desired elements into geometrically convenient subelements with hanging nodes and then it eliminates the hanging nodes by forcing refinement of additional elements. This is illustrated in Figure 3.8. This approach preserves regularity of the mesh but in the case of triangles, it introduces elements with

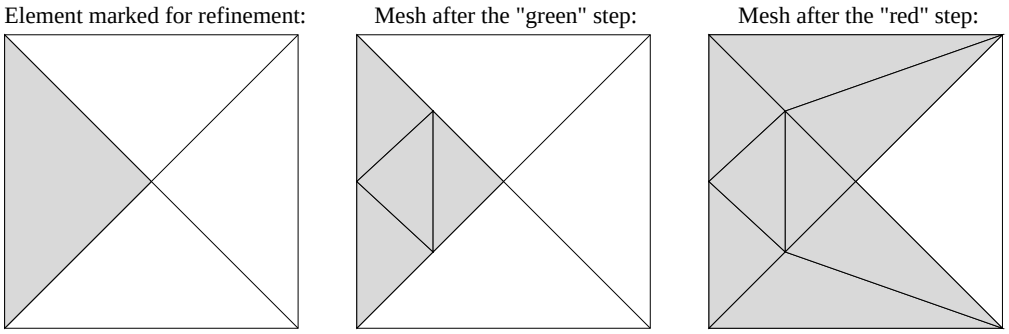

Figure 3.8: Red-green refinement.

sharp angles which are not desirable in the finite element analysis. In the case of quadrilaterals, it also negatively influence the element shapes. It becomes extremely cumbersome when repeated refinements occur in the same part of the domain (e.g., toward a boundary layer or point singularity).

The "red" refinements can be avoided by introducing *hanging nodes*, i.e., by allowing *irregular meshes* where element vertices lie in the interior of edges of other elements. In order to keep the computer implementation simple, most finite element working with hanging nodes limit the maximum difference of refinement levels of adjacent elements to one (so-called 1-*irregularity rule*) – see, e.g., [17, 18, 20]. In the following, by k-*irregularity rule* (or *hanging nodes of level* k) we mean this type of restriction where the maximum difference of refinement levels of adjacent elements is k. In this context, $k = 0$ corresponds to adaptivity with regular meshes and $k = \infty$ to adaptivity with arbitrary-level hanging nodes.

It is illustrated in Figure 3.9 that even the 1-irregularity rule does not avoid all forced refinements:

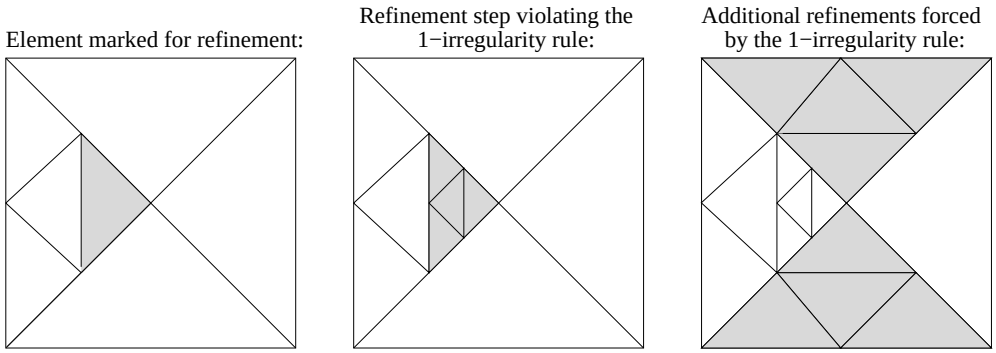

Figure 3.9: Refinement with 1-irregularity rule.

The amount of forced refinements in the mesh depends strongly on the level of hanging nodes. Next we introduce a model problem and show that the level of hanging nodes also influences significantly both the number of degrees of freedom and condition number of stiffness matrices.

Model problem Consider a Poisson equation $-\Delta u = f$ in Ω with $u = 0$ on the boundary of Ω, where $\Omega = (-1, 1)^2$. Assume a right-hand side f such that the corresponding exact solution u is zero everywhere in Ω with the exception of a local perturbation occuring inside of a small triangle T_n with the vertices $[-2^{-n}, -2^{-n}]$, $[0, 0]$, $[-2^{-n}, 2^{-n}]$. The domain is covered with a coarse four-element mesh shown in Figure 3.10. For simplicity, all elements are equipped with a uniform polynomial degree $p \geq 1$.

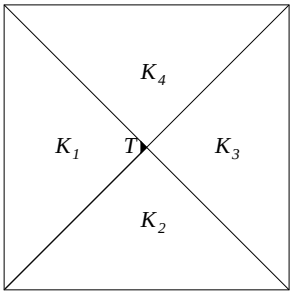

Figure 3.10: Domain and initial mesh for the study of k-irregularity rules.

We perform the following simple experiment: Starting from the coarse mesh shown in Figure 3.10, we run an adaptive algorithm which in each step applies the "green" refinement to every mesh triangle K such that $K \cap T_n \neq \emptyset$ and $K \not\subset T_n$. Figure 3.11 shows final meshes obtained with various levels of hanging nodes for $n = 5$:

Notice that, since $T_n \subset K_1$, all refinements within the elements K_2, K_3, K_4 are forced by regularity requirements and they do not improve the resolution.

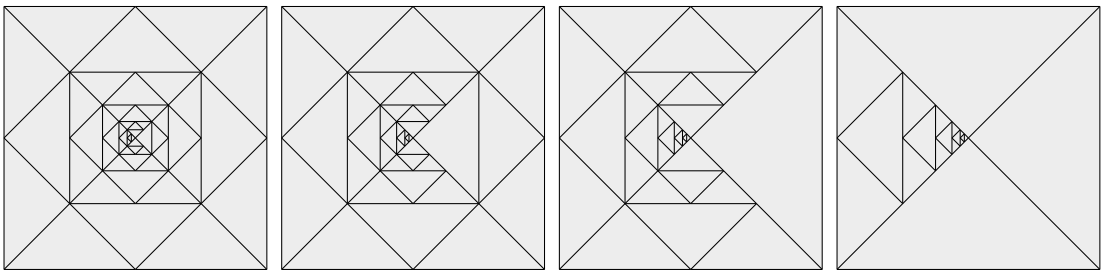

Figure 3.11: Meshes obtained with k-irregularity rules, $k = 1, 2, 3, \infty$.

The reader can see that the amount of forced refinements decreases as k grows, vanishing completely with $k = \infty$.

Next let us choose, e.g., $p = 3$, and run the same adaptive procedure for $n = 1, 2, \ldots, 15$. Figure 3.12 shows the number of degrees of freedom corresponding to the final meshes. The horizontal axis represents the spatial scale 2^{-n}. For the same computations, Figure 3.13 shows the condition number of the corresponding stiffness matrices.

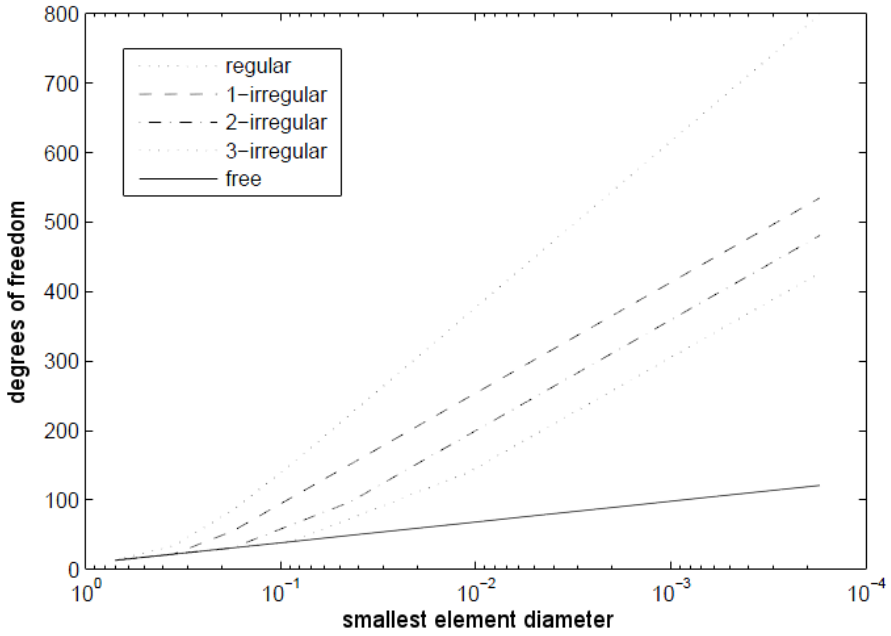

Figure 3.12: Size of the stiffness matrix for final meshes. Level of hanging nodes: $k = 0, 1, 2, 3, \infty$.

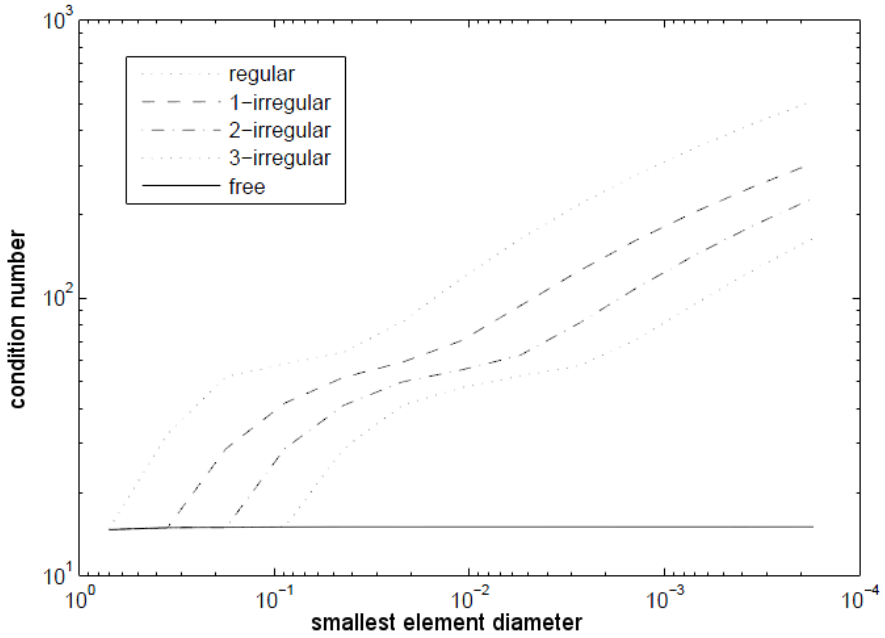

Figure 3.13: Condition number of the stiffness matrix for final meshes. Level of hanging nodes: $k = 0, 1, 2, 3, \infty$.

3.3 Data structures for *hp*-adaptivity

The need for mesh adaptivity brings further demands that must be taken into account in the design of the data structures. A tree-like hierarchy of element refinement levels must be introduced and also more complicated neighbor search algorithms are required to correctly initialize the edge and vertex nodes and the integration points on the edges. The complexity of the data structures reflects strongly on the complexity of the rest of the finite element code.

Simplifying assumptions, such as the k-irregularity rules, are often imposed on the mesh with the aim of reducing code complexity. However, these assumptions can deteriorate the numerical performance of the code.

In the following we present an original design of data structures for adaptive meshes, free of any regularity assumptions yet retaining great simplicity.

Calculation of fluxes in DGFEM with arbitrary-level hanging will also be described.

3.3.1 Node and element structures

It should be noted first that we have departed from traditional mesh data structures storing complete information about the finite element problem. Our mesh structures contain geometrical information only. The remaining data, including basis function indices, boundary conditions, polynomial degrees, etc., are stored in separate structures describing concrete finite element spaces (H^1, $\boldsymbol{H}$(curl), ...) and are accessible via the identification numbers of nodes and elements. This was done to allow multiple spaces and multiple element types to co-exist on the same mesh, which is vital for solving multi-physics and coupled problems.

The entire mesh is defined by two arrays of the following structures. The first one stores all nodes:

```
structure Node
{
  // Type of the Node - either vertex, or edge.
  type;

  // Physical coordinates of this Node, in case it is a vertex Node.
  coordinate x;
  coordinate y;

  // Two neighboring elements of this Node, in case it is an edge Node.
  Element elements[2];
};
```

This structure defines both vertices and edges. The standard vertex positions `x`, `y` are placed in the vertex variant of the `Node` structure. The edge variant contains pointers to the two elements sharing the edge node (useful e.g. when looking for a neighbor of an element sharing a particular edge in order to calculate the numerical flux across the edge).

The attribute `type` determines the variant of the structure. The remaining attributes are omitted for simplicity.

Elements are defined by the second structure:

```
structure Element
{
  // Type of the Element - either triangle, or quadrilateral.
  type;

  // True or false whether this element has or has not been refined further.
  Boolean refined;

  // Vertex nodes
  Node vn[4];

  // Edge nodes
  Node en[4];
};
```

We also keep elements that have been refined further, for purpose of finding neighors, adaptivity, and derefinement of the meshes which is important for time-dependent problems.

Elements that are not refined store pointers to appropriate vertices and edges.

Elements that have been refined only store information about elements into which they were refined for the above reasons.

Triangular and quadrilateral elements share the same structure and are distinguished by the attribute `type`.

3.3.2 Eliminating neighbor search by hashing

To properly initialize edges (edge variant of the `Node` structure) after reading a mesh file, one has to construct neighbor lists for all elements and use them in such a way that only one `Node` is created for each edge. Further problems arise when certain elements are refined after automatic mesh adaptation. Unless hanging nodes are removed by extra refinements, no longer is each edge shared by at most two elements (there can be more of them). Standard neighbor lists fail to fully capture such situations and thus more complex data structures, e.g. trees [4], have to be introduced.

We have avoided all of these problems by introducing a function which, given two vertices, returns the vertex halfway between them. If no such vertex exists, it is created first.

Details of the approach can be found in Appendix 1.

3.3.3 Finding all neighbors of an element

In element-wise assembling procedure with arbitrary-level hanging nodes, when calculating fluxes across mesh edges, one needs values from both sides of the edge. To achieve this, the integration points need to be matched correctly, and proper

values in these points from the adjacent element need to be extracted for all involved functions defined on the mesh (basis functions, test functions, previous time level solutions). This is not easy to manage. A simplified version of the algorithm is in Appendix 1.

3.4 Automatic *hp*-adaptivity

The major difference between adaptivity in standard FEM and adaptivity in *hp*-FEM is the large number of element refinement options in the latter case. In standard *h*-adaptivity, elements can only be subdivided in space, e.g., using the *red-green* refinement technique (see [1] and the references therein). In *hp*-adaptivity, one can either increase the polynomial degree of elements without subdividing them in space or one can split elements in space and distribute the polynomial degree on the subelements in multiple ways.

3.4.1 Reference solution

The presence of many *hp*-refinement options per element implies that classical error estimates (in the form of one number per element) do not provide enough information to guide *hp*-adaptivity. Instead, one needs to use some information about the *shape* of the error function $e_{h,p} = u - u_{h,p}$. In principle, this information could be obtained by estimating higher derivatives, but such approach is not very practical. Usually it is more convenient to use a *reference solution*, i.e., an approximation u_{ref} which is at least one order more accurate than the coarse mesh solution $u_{h,p}$. The *hp*-adaptivity is then guided by an a-posteriori error estimate of the form

$$e_{h,p} = u_{ref} - u_{h,p} \tag{3.3}$$

Such approach, as opposed to classical error estimates, is virtually equation-independent. In this work we are constructing the enriched finite element space by uniform subdivision of all mesh elements and by increasing all element polynomial degrees by one, i.e.

$$u_{ref} = u_{h/2,p+1}$$

The reference solution u_{ref} can be obtained efficiently by utilizing information about lower frequencies from $u_{h,p}$ [4, 20, 21] if iterative solvers are used or by reconstructing the LU decomposition of the stiffness matrix from the LU-decomposed coarse mesh stiffness matrix when using direct sparse solvers, e.g. UMFPACK. However, we have not attempted any such optimization in this work, even though we admit that this issue must be addressed in order for the methods to be usable in practice.

3.4.2 Outline of the algorithm

The outer loop of our adaptivity algorithm is formally similar to the one of L. Demkowicz et al. (see [5]), with minor differences. The outline of the algorithm is as follows:

A) Assume an initial coarse mesh $\mathcal{T}_h$ and an initial space V_h consisting of piecewise-constant, linear or quadratic elements. User input includes:

(a) prescribed relative tolerance $TOL > 0$ for the rate of a suitable norm of the approximate error function (3.3) and the norm of the approximate solution

(b) threshold ERRT specifying when to stop refining when refining elements sequentially from the one with the largest values of the approximate error function in each *hp*-adaptivity step.

B) User input may also include:

(a) maximum increase of degrees of freedom MAX_STEP_NDOF in each adaptivity step
OR

(b) maximum number of degrees of freedom MAX_NDOF.

C) Create a fine mesh, $\mathcal{T}_{h/2} = \{K_1, K_2, \ldots, K_{M_ref}\}$.

D) Create a reference space, $V_{h/2,p+1} := \{v \in L^2(\Omega);\ v|_{K_m} \in P^{p_m}(K_m) + 1 \text{ for all } 1 \leq m \leq M_ref\}$, where the polynomial degree $p_m(K_m)$ is inherited from corresponding elements of the coarse mesh.

E) Compute fine mesh approximation $u_{h/2,p+1} \in V_{h/2,p+1}$ on $\mathcal{T}_{h/2}$.

F) Project the fine mesh approximation onto the coarse space V_h. In this way, obtain a coarse mesh approximation.

G) Calculate the desired norm (H^1 norm, H^1 seminorm, L^2 norm, custom norm) of the fine mesh approximations $NORM_i$ on every element K_i in the mesh, $i = 1, 2, \ldots, M$. Construct the approximate error function (3.3) as the difference between the fine and coarse mesh approximations, calculate its energy norm ERR_i on every element K_i in the mesh, $i = 1, 2, \ldots, M$. Calculate the global error estimate

$$ERR = \left(\sum_{i=1}^{M} ERR_i \right) / \left(\sum_{i=1}^{M} NORM_i \right).$$

H) If $ERR \leq TOL$, stop computation and proceed to postprocessing.

I) Sort all elements into a list L according to their value $ERR_i/NORM_i$ in decreasing order.

J) Determine the maximum of element errors, $ERR_{max} = \max\{ERR_i/NORM_i\}$, by taking the first item of L.

K) Let $NDOF$ be the current number of degrees of freedom of V_h. Repeat the following cycle:

   (a) If MAX_STEP_NDOF is set, and if the already added degrees of freedom in this hp-adaptivity step is greater than MAX_STEP_NDOF, break the cycle

   (b) Take the next element K_i from the list L

   (c) If $ERR_i/NORM_i < ERRT \cdot ERR_{max}$, break the cycle

   (d) Perform hp-refinement of K_i (to be described in more detail below).

   (e) If MAX_NDOF is set, and if the total number of degrees of freedom is now greater than MAX_NDOF, end the cycle.

L) Continue with step C.

The crucial issue in the outer loop is determining how many elements should be refined. In [5] all elements whose error ERR is greater than 70% of the maximum error ERR_{max} are taken (ie. $ERRT = 0.7$). This is a relatively conservative choice which leads to good convergence curves, but may result in too many steps of the algorithm, which should be avoided as the evaluation of u_{ref} is very expensive. Our experience shows that setting $ERRT$ as low as 15% results in equally good convergence curves for most problems, but achieved in much fewer steps and in turn in much shorter time. Occasionally, however, the low value of $ERRT$ may cause too many elements to be refined and thus we introduced the limit MAX_STEP_NDOF of the number of degrees of freedom added in each step. Typically, one should not increase the number of degrees of freedom in each step by a large number.

3.4.3 Selecting optimal hp-refinement

Some implementation of the algorithm for determining optimal hp-refinements, such as in [5], are quite complex. Built upon the projection-based interpolation theory, such approach first finds optimal hp-refinement of mesh edges using a 1D version of the hp-adaptive algorithm. The edge refinements then determine h-refinement of elements. Finally, best polynomial degrees for element interiors are selected. Each step of the algorithm represents a considerable implementation burden.

We have implemented a much simpler scheme, which is local in the sense that element refinements are selected without regard to the refinements of neighboring elements. For all elements $K \in \mathcal{T}_h$ of polynomial degree p picked by the outer loop we consider the following $N = 83$ hp-refinement options:

A) Increase the polynomial degree of K to $p + 1$ without spatial subdivision.

B) Increase the polynomial degree of K to $p+2$ without spatial subdivision.

C) Split K into four similar triangles K_1, K_2, K_3, K_4. Define p_0 to be the integer part of $(p+1)/2$. For each K_i, $1 \leq i \leq 4$ consider polynomial degrees $p_0 \leq p_i \leq p_0 + 2$.

For each of these N options we perform a standard e.g. L_2-projection of the reference solution u_{ref} onto the corresponding polynomial space. The candidate with smallest projection error is selected for hp-refinement of the element K.

Each of the N projection problems requires the solution of a small system of linear algebraic equations with a symmetric positive-definite matrix. The solution of these systems can be further optimized by exploiting the incremental nature of the Cholesky decomposition algorithm and the fact that the spaces in item 3 above are partially embedded.

Example We again solve the model problem (2.1) with the same settings as in Paragraph 2.1. This time, automatic hp-adaptivity will be used. Although the maximum value of the exact solution is 1.0, due to the presence of undershoots and overshoots, the maximum values here are higher, as was seen already in section 2.4. This can be avoided by using a suitable shock capturing method, as it is done in the last chapter with numerical examples for the Euler equations.

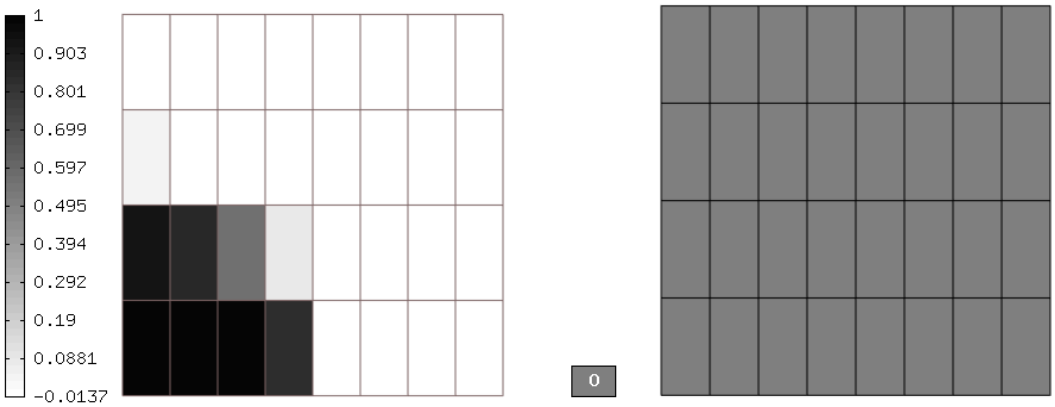

Figure 3.14: Solution and a mesh, relative error between the fine mesh and the coarse mesh approximation: 25.7746%.

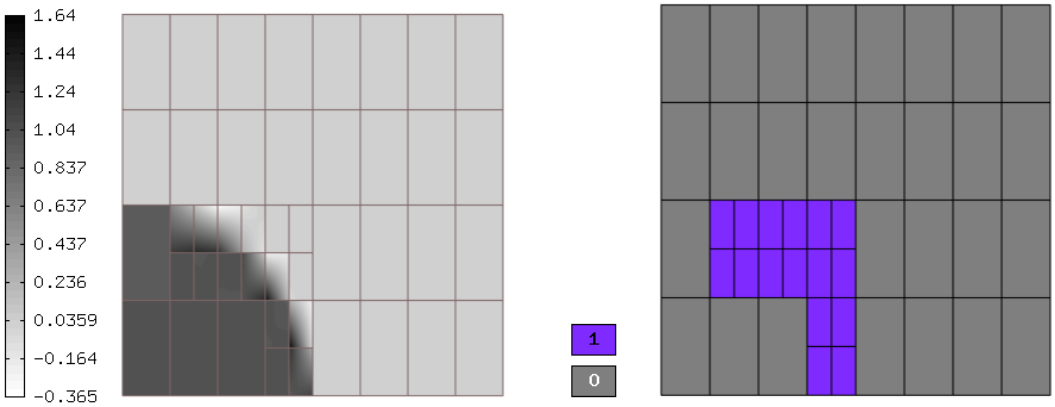

Figure 3.15: Solution and a mesh, relative L^2-norm difference between the fine mesh and the coarse mesh approximation: 10.3565%.

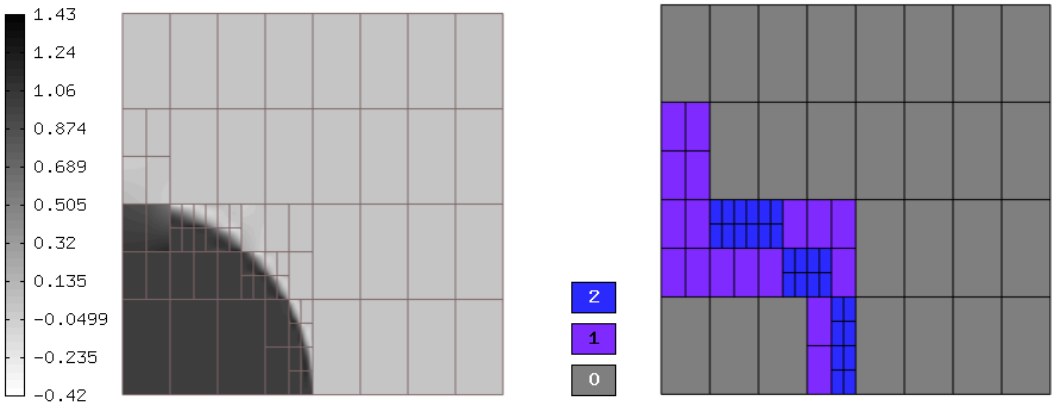

Figure 3.16: Solution and a mesh, relative L^2-norm difference between the fine mesh and the coarse mesh approximation: 5.58551%.

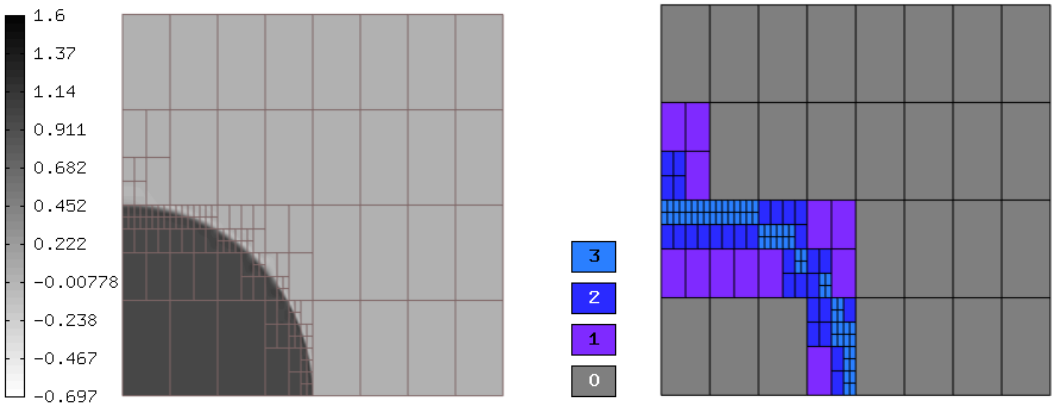

Figure 3.17: Solution and a mesh, relative L^2-norm difference between the fine mesh and the coarse mesh approximation: 2.43584%.

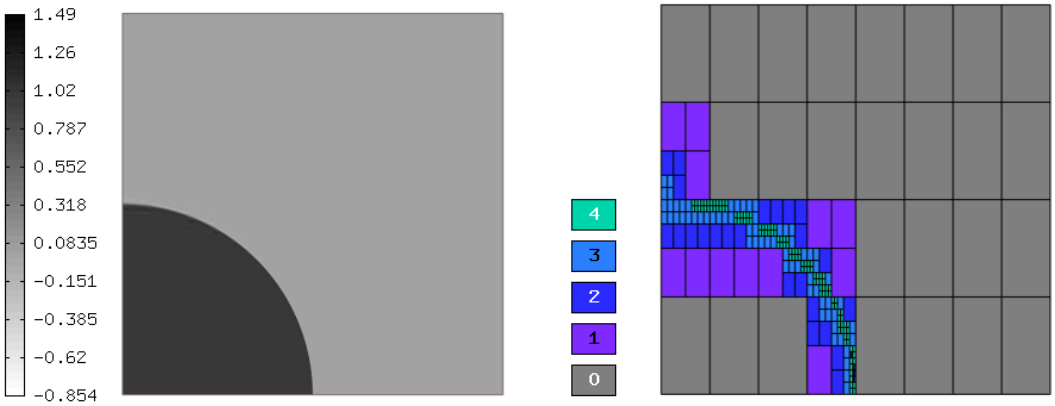

Figure 3.18: Solution and a mesh, relative L^2-norm difference between the fine mesh and the coarse mesh approximation: 1.27544%.

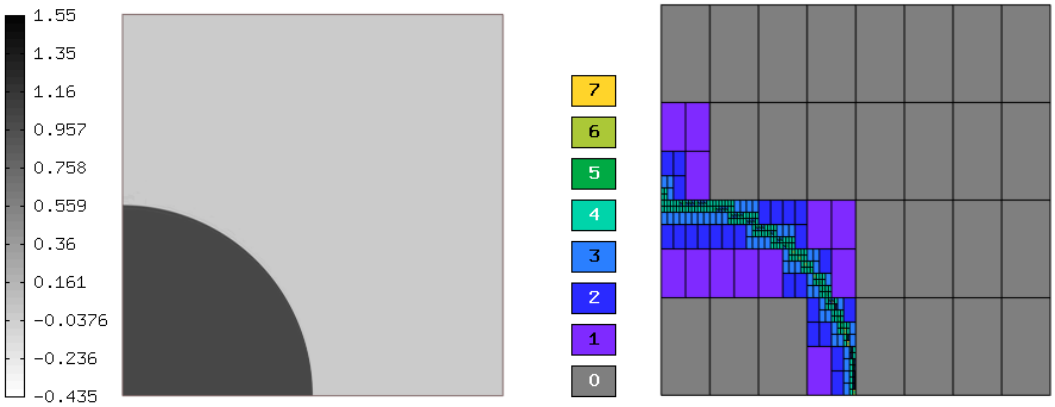

Figure 3.19: Solution and a mesh, relative L^2-norm difference between the fine mesh and the coarse mesh approximation: 0.919277%.

3.4.4 Automatic *hp*-adaptivity for time-dependent problems

In the previous chapters we were concerned with the stationary problems. Let us now discuss automatic adaptivity for transient problems. Time-dependent coupled problems are challenging since one has to capture transient phenomena with sufficient accuracy, while the size of the problem must stay reasonably small. This leads to an obvious need for dynamically changing meshes for such problems. In most scientific computations, where dynamical meshes are used for time-dependent problems, data-transfer methods are necessary to move solution values between meshes for different time levels. Data-transfer methods can cause additional error and in case of time-dependent problems, repeated transfers (usually simple interpolation) between meshes can have disastrous consequences.

We use orthogonal projections of the solution of transient problems using dynamical *hp*-meshes obtained fully automatically by the *hp*-adaptive algorithm. With the automatic *hp*-adaptivity and dynamical meshes the question of coarsening meshes between time levels arises. Mesh derefinement is particularly important in problems where sharp fronts (internal layers) move through the domain leaving smooth solutions behind them. We propose an original coarsening algorithm suitable for *hp*-FEM based on the *super-coarse solution*, which results in substantially fewer adaptive iterations.

The adaptive *hp*-FEM algorithm for time-dependent problems we use is obtained by combining the classical Rothe's method for time discretization with adaptive *hp*-FEM for the space discretization. The Rothe's method is a natural counterpart of the widely used Method of Lines (MOL). Recall that the MOL performs discretization in space while keeping the time variable continuous, which leads to a system of ODEs in time. The Rothe's method, on the contrary, preserves the continuity of the spatial variable while discretizing time. In every time step, an evolutionary PDE is approximated by means of one or more time-independent ones. For one step methods (such as implicit Runge-Kutta), the number of the time-independent equations per time step is proportional to the order of accuracy of the time discretization method. For example, when employing the implicit Euler method, one has to solve just one time-independent PDE per time step:

$$\frac{\partial u}{\partial t} = F(t,u) \quad \Rightarrow \quad \frac{u^{n+1}-u^n}{\Delta t} = F(t^{n+1},u^{n+1}) \tag{3.4}$$

The Rothe's method is similar to the MOL if no adaptivity in space or time takes place, but it provides a better setting for the application of spatially adaptive algorithms. The spatial discretization error can be controlled by solving the time-independent equations adaptively, and the size of the time step can be adjusted using standard ODE techniques [3].

In this subsection, first the notion of *dynamical meshes* will be presented, then the refining and coarsening algorithms for time dependent problems are shown.

Dynamical meshes By $\mathcal{T}_m$ let us denote a uniform coarse mesh covering the computational domain Ω. This mesh (called *master mesh*) is shared by the solution at all time levels, in other words all meshes can be obtained from this mesh by elementary refinements. At each time instant t_n an optimal mesh is found to suit the best the solution $\boldsymbol{u}^n(\boldsymbol{x})$. On the $(n+1)$st time level, the approximated solution $\boldsymbol{u}^n(\boldsymbol{x})$, that has been obtained in the previous time step, is used as data. Note, however, that $\boldsymbol{u}^n$ is defined on a locally refined mesh that was created automatically during the nth time step, while the unknown $\boldsymbol{u}^{n+1}$ is solved adaptively starting from a coarser mesh. As a result, the meshes obtained on each time level are different, i.e., the mesh changes dynamically in time.

As mentioned above, in transient problems the optimal meshes are on each time level obtained automatically and they can change from one time step to another, which induces need for both refining and coarsening strategies.

Refining algorithm

In the previous the goal of the adaptive algorithm was to decrease the error as low as possible. On the other hand for time-dependent problems we want to sustain the space error on approximately the same level, which would result into meshes with smoothly changing number of degrees of freedom. For time-dependent problems, as well as for time-independent, the amount of elements refined in one adaptive iteration will depend on the global solution error estimate in that iteration. As described, one can drive the refining algorithm by variables TOL, $ERRT$, MAX_STEP_NDOF, MAX_NDOF.

Coarsening algorithm

The most primitive strategy to coarsen the mesh on the next time level is to start on each level from the very coarse (master) mesh and perform the automatic adaptivity to find the optimal mesh for a particular solution $\boldsymbol{u}^n(\boldsymbol{x})$. Although this would result in the optimal meshes in each iteration, it is virtually impossible to carry it out due to the immense computational time demands. Moreover, a lot of work would be wasted since the solutions from two adjacent time steps usually differ only mildly even though the solution changes significantly in the whole time domain. Global derefinement would result in similar difficulties.

We present a coarsening algorithm that prepares the mesh for the adaptive algorithm on the next time level by local derefinements. Thus, we remove only unnecessary refinements from previous time levels. In this way the meshes for particular time steps are suboptimal, but the number of adaptive iterations performed in each time step significantly decreases.

In higher-order finite element method we have two questions. First question is which elements can be coarsened and second is how to coarsen them. In *hp*-FEM we can either decrease the polynomial order of the element or derefine the element

geometrically. We can also employ various choices for assigned polynomial degree. Some possible choices are in Figure 3.20.

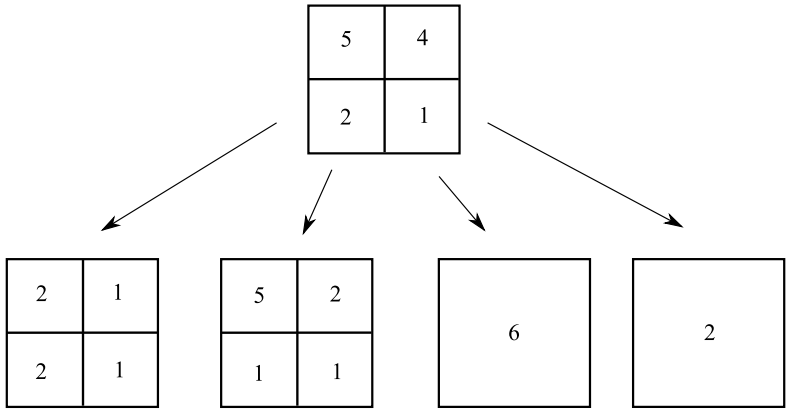

Figure 3.20: Coarsening choices for an element.

However, the coarsening serves only to prepare the mesh for the next adaptive process, thus we do not have to search for the best unrefinement as we do in case of refinements. It is perfectly sufficient to perform any coarsening that does not cause error increase and the mesh will optimize itself in the next adaptive loop. To keep the algorithm reasonably simple we allow only coarsening as reversing of previous element refinements. In this way the tree-like structure of meshes starting from the *master mesh* is preserved. Unfortunately, the coarsening algorithm cannot be just an opposite to the refining. While with the refining algorithm we are seeking for the best candidate in the sense of error decrease, here we are looking for a candidate whose error will not exceed some tolerance (so that it would not be subsequently refined).

Let us denote by e_{max} the error of the element with maximal error from the last adaptive iteration. From Alg. 3.4.2 we know that errors of all elements are below ERR_{max} and that it should be satisfied as well after the coarsening procedure. First we calculate so called *super-coarse solution* for polynomial orders. By that we mean that polynomial order on all elements is decreased by one. We calculate error estimates for all elements in the super-coarse mesh with respect to the reference solution and lower the polynomial order on those elements whose error after coarsening is less than $k * ERR_{max}$, where parameter $k \in (0, 1)$ is chosen to ensure that element will not be subsequently refined in the adaptive procedure on the next time level. Similar procedure is run for a spatial coarsening – *super-coarse solution* is calculated on the mesh where all elements with four active subelements are coarsened and polynomial order on such elements is assigned to be the maximum of orders on four subelements. In a similar way as before we determine which elements can be also spatially unrefined without significant increase of the error.

The whole procedure for time-dependent problems with the refining and coars-

ening strategy is described in Algorithm 1. Effectivity of the approach is demonstrated on the numerical examples in the last chapter of this thesis.

Algorithm 1: Adaptive algorithm for time-dependent problems with improved stopping criterion.

```
foreach time level do
    repeat
        compute reference solution u_ref;
        project to get a solution on current mesh u;
        evaluate global error estimate ERR_gl and local errors on elements
        ERR_i;
        if ERR_gl < TOL then
            done = true;
            break;
        else
            sort all elements by their error;
            processed error = 0;
            foreach element do
                if (processed error > c * (ERR_gl - TOL)) or
                (ERR_i < k * ERR_max) then
                    done = true;
                    break;
                else
                    find optimal refinement;
                    processed error += ERR_i;
    until not done;
    calculate super-coarse solution for polynomial orders;
    decrease polynomial orders when possible;
    calculate super-coarse solution for spatial refinements;
    geometrically coarsen elements when possible;
```

Chapter 4

Dicontinuous Galerkin discretization of the Euler equations

4.0.5 Discontinuous Galerkin space semidiscretization

The approximate solution will be sought at each time instant t as an element of the finite-dimensional space

$$[V_h]^4, \tag{4.1}$$

where V_h is defined in Section 3.1.1. Functions $\boldsymbol{\varphi}_h \in [V_h]^4$ are in general discontinuous on interfaces Γ_{ij}.

By $\boldsymbol{\varphi}_h|_{\Gamma_{ij}}$ and $\boldsymbol{\varphi}_h|_{\Gamma_{ji}}$ we denote the values of $\boldsymbol{\varphi}_h$ on Γ_{ij} considered from the interior and the exterior of K_i, respectively. The symbols

$$\langle \boldsymbol{\varphi}_h \rangle_{ij} = \frac{1}{2}\left(\boldsymbol{\varphi}_h|_{\Gamma_{ij}} + \boldsymbol{\varphi}_h|_{\Gamma_{ji}}\right), \quad [\boldsymbol{\varphi}_h]_{ij} = \boldsymbol{\varphi}_h|_{\Gamma_{ij}} - \boldsymbol{\varphi}_h|_{\Gamma_{ji}} \tag{4.2}$$

denote the average and jump of a function $\boldsymbol{\varphi}_h$ on $\Gamma_{ij} = \Gamma_{ji}$.

In order to derive the discrete problem, we multiply (1.73) by a test function $\boldsymbol{\varphi}_h \in \boldsymbol{S}_h$, integrate over any element K_i, $i \in I$, apply Green's theorem and sum over all $i \in I$. Then we approximate fluxes through the faces Γ_{ij} with the aid of a numerical flux $\boldsymbol{H} = \boldsymbol{H}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{n})$ in the form

$$\int_{\Gamma_{ij}} \sum_{s=1}^{2} \boldsymbol{f}_s(\boldsymbol{w}(t))\,(n_{ij})_s \cdot \boldsymbol{\varphi}_h \, dS \approx \int_{\Gamma_{ij}} \boldsymbol{H}(\boldsymbol{w}_h(t)|_{\Gamma_{ij}}, \boldsymbol{w}_h(t)|_{\Gamma_{ji}}, \boldsymbol{n}_{ij}) \cdot \boldsymbol{\varphi}_h \, dS. \tag{4.3}$$

If we introduce the forms

$$(\boldsymbol{w}_h, \boldsymbol{\varphi}_h)_h = \int_{\Omega_h} \boldsymbol{w}_h \cdot \boldsymbol{\varphi}_h \frac{d\boldsymbol{x}}{d,} \tag{4.4}$$

$$\begin{aligned} \tilde{b}_h(\boldsymbol{w}_h, \boldsymbol{\varphi}_h) &= -\sum_{K\in T_h} \int_K \sum_{s=1}^{2} \boldsymbol{f}_s(\boldsymbol{w}_h) \cdot \frac{\partial \boldsymbol{\varphi}_h}{\partial x_s} \frac{d\boldsymbol{x}}{d,} \\ &+ \sum_{K_i\in T_h} \sum_{j\in S(i)} \int_{\Gamma_{ij}} \boldsymbol{H}(\boldsymbol{w}_h|_{\Gamma_{ij}}, \boldsymbol{w}_h|_{\Gamma_{ji}}, \boldsymbol{n}_{ij}) \cdot \boldsymbol{\varphi}_h \frac{dS}{d,} \end{aligned} \tag{4.5}$$

we can define an *approximate solution* of (1.73) as a function $\boldsymbol{w}_h$ satisfying the conditions

$$\begin{aligned} &\text{(a)} \quad \boldsymbol{w}_h \in C^1([0,T]; \boldsymbol{S}_h), \\ &\text{(b)} \quad \frac{d}{dt}(\boldsymbol{w}_h(t), \boldsymbol{\varphi}_h)_h + \tilde{b}_h(\boldsymbol{w}_h(t), \boldsymbol{\varphi}_h) = 0 \\ &\qquad \forall\, \boldsymbol{\varphi}_h \in \boldsymbol{S}_h \quad \forall\, t \in (0,T), \\ &\text{(c)} \quad \boldsymbol{w}_h(0) = \Pi_h \boldsymbol{w}^0, \end{aligned} \tag{4.6}$$

where $\Pi_h \boldsymbol{w}^0$ is the L^2-projeftion of the initial condition $\boldsymbol{w}^0$ onto the space $[V_h]^4$.

4.1 Numerical Fluxes

The DGFEM discretization of the Euler equations in their conservative form follows the principles of the section 2.4. We take the equations in the form (1.73). We leave the time derivative for now, and we need to find suitable numerical fluxes to approximate the fluxes $\partial \boldsymbol{f}_x, \partial \boldsymbol{f}_y$ through the faces Γ_{ij}. The numerical fluxes will be sought so that the requirements from the subsection 2.2.2 are met.

Construction of some numerical fluxes The following type of numerical fluxes are usually called *flux vector splitting schemes* and we use the knowledge from the section 1.2, namely (1.81), (1.83), ..., and (1.102). On the basis of (1.102) we define the matrices

$$\Lambda^{\pm} = \operatorname{diag}\left(\lambda_1^{\pm}, ..., \lambda_m^{\pm}\right), \quad |\Lambda| = \operatorname{diag}\left(|\lambda_1|, ..., |\lambda_m|\right). \tag{4.7}$$

Also we define

$$\mathbb{P}^{\pm} = \mathbb{P}\Lambda^{\pm}\mathbb{P}^{-1}, \quad |\mathbb{P}| = \mathbb{P}|\Lambda|\mathbb{P}^{-1}. \tag{4.8}$$

These matrices depend on $\boldsymbol{w} \in D$ and $\boldsymbol{n} \in \mathcal{S}_1$. Now we define the following two schemes that will later be used in the numerical examples.

A) *The Steger-Warming scheme* has the numerical flux:

$$\boldsymbol{H}_{SW}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{n}) = \mathbb{P}^{+}(\boldsymbol{u},\boldsymbol{n})\,\boldsymbol{u} + \mathbb{P}^{-}(\boldsymbol{v},\boldsymbol{n})\,\boldsymbol{v}, \quad \boldsymbol{u},\boldsymbol{v} \in D,\ \boldsymbol{n} \in \mathcal{S}_1.$$

This scheme is rather diffusive, so another scheme is preferred.

B) *The Vijayasundaram scheme* has the numerical flux:

$$\boldsymbol{H}_{V}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{n}) = \mathbb{P}^{+}\left(\frac{\boldsymbol{u}+\boldsymbol{v}}{2},\boldsymbol{n}\right)\boldsymbol{u} + \mathbb{P}^{-}\left(\frac{\boldsymbol{u}+\boldsymbol{v}}{2},\boldsymbol{n}\right)\boldsymbol{v}, \quad \boldsymbol{u},\boldsymbol{v} \in D,\ \boldsymbol{n} \in \mathcal{S}_1.$$

This scheme contains a *partial upwinding*, where the flux is computed at the point $x_{i+\frac{1}{2}}$ $\left(x_{i-\frac{1}{2}}\right)$ with the use of the value $\boldsymbol{w}_i^k$ or $\boldsymbol{w}_{i+1}^k$ $\left(\boldsymbol{w}_{i-1}^k \text{ or } \boldsymbol{w}_i^k\right)$ corresponding to the mesh point located in the upwind direction wrt. the propagation speed given by the eigenvalues λ_i. In the Steger-Warming scheme we speak of a *full upwinding*.

4.2 Time discretization

This section closely follows [6]. Using numerical fluxes we can introduce new forms

$$\begin{aligned}
(\boldsymbol{w}_h,\boldsymbol{\varphi}_h) &= \int_\Omega \boldsymbol{w}_h(\boldsymbol{x})\cdot\boldsymbol{\varphi}_h(\boldsymbol{x})\,d\boldsymbol{x},\\
\tilde{b}_h(\boldsymbol{w}_h,\boldsymbol{\varphi}_h) &= -\sum_{K\in\mathcal{T}_h}\int_K \sum_{s=1}^{2} \boldsymbol{f}_s(\boldsymbol{w}_h(\boldsymbol{x}))\cdot\frac{\partial\boldsymbol{\varphi}_h(\boldsymbol{x})}{\partial x_s}d\boldsymbol{x} \qquad (4.9)\\
&+ \sum_{K_i\in\mathcal{T}_h}\sum_{j\in\mathcal{S}(i)}\int_{\Gamma_{ij}} \boldsymbol{H}\left(\boldsymbol{w}(t)|_{\Gamma_{ij}},\boldsymbol{w}(\boldsymbol{x})|_{\Gamma_{ji}},\boldsymbol{n}_{ij}\right)\cdot\boldsymbol{\varphi}_h(\boldsymbol{x})\,dS
\end{aligned}$$

for $\boldsymbol{w}_h,\boldsymbol{\varphi}_h \in [V_h]^4$. We say that $\boldsymbol{w}_h$ is the approximate solution of (1.73) in $Q_T = \Omega\times(0,T)$, if it satisfies the conditions

A)

$$\boldsymbol{w}_h \in \mathcal{C}^1\left([0,T],[V_h]^4\right) \qquad (4.10)$$

B)

$$\frac{d}{dt}(\boldsymbol{w}(t),\boldsymbol{\varphi}_h) + \tilde{b}_h(\boldsymbol{w}_h(t),\boldsymbol{\varphi}_h) = 0 \quad \forall\boldsymbol{\varphi}_h \in [V_h]^4\,\forall t\in(0,T) \qquad (4.11)$$

C)

$$\boldsymbol{w}_h(0) = \Pi_h\boldsymbol{w}^0, \qquad (4.12)$$

Relations (4.11) represent a system of ordinary differential equations which can be solved by a suitable numerical method. Since we are interested in applying the Rothe's method, we now want to discretize the time derivative. In order to do so, we consider a partition $0 = t_0 < t_1 < t_2 < ...$ of the time interval $(0, T)$ and set $\tau_k = t_{k+1} - t_k$. We use the notation $\boldsymbol{w}_h^k$ for the approximation of $\boldsymbol{w}_h(t_k)$. Then we apply the simple implicit *backward Euler method* and our *discrete problem* reads: for each $k \geq 0$ find $\boldsymbol{w}_h^{k+1}$ such that

A)

$$\boldsymbol{w}_h^{k+1} \in [V_h]^4 \tag{4.13}$$

B)

$$\left(\frac{\boldsymbol{w}_h^{k+1} - w_h^k}{\tau_k}, \boldsymbol{\varphi}_h\right) + \tilde{b}_h\left(\boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) = 0 \quad \forall \boldsymbol{\varphi}_h \in [V_h]^4, \ k = 0, 1, ... \tag{4.14}$$

C)

$$\boldsymbol{w}_h^0 = \Pi_h \boldsymbol{w}^0. \tag{4.15}$$

This scheme leads to a system of highly nonlinear algebraic equations whose numerical solution is rather complicated. In order to simplify the problem, in the following we shall linearize relations (4.14) and obtain a linear system.

4.2.1 Linearization

By (4.9), for $\boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h \in [V_h]^4$ we have

$$\tilde{b}_h\left(\boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) = -\sum_{K \in \mathcal{T}_h} \int_K \sum_{s=1}^{2} \boldsymbol{f}_s\left(\boldsymbol{w}_h^{k+1}(\boldsymbol{x})\right) \cdot \frac{\partial \boldsymbol{\varphi}(\boldsymbol{x})}{\partial x_s} d\boldsymbol{x} \quad (=: \tilde{\sigma}_1) \tag{4.16}$$

$$+ \sum_{K_i \in \mathcal{T}_h} \sum_{j \in S(i)} \int_{\Gamma_{ij}} \boldsymbol{H}\left(\boldsymbol{w}_h^{k+1}|_{\Gamma_{ij}}, \boldsymbol{w}_h^{k+1}|_{\Gamma_{ji}}, \boldsymbol{n}_{ij}\right) \cdot \boldsymbol{\varphi}_h(\boldsymbol{x})\, dS \quad (=: \tilde{\sigma}_2) \tag{4.17}$$

The terms $\tilde{\sigma}_1$, and $\tilde{\sigma}_2$ are linearized as follows. We set

$$\tilde{\sigma}_1 \approx \sigma_1 = \sum_{K \in \mathcal{T}_h} \int_K \sum_{s=1}^{2} \mathbb{A}_s\left(\boldsymbol{w}_h^k(\boldsymbol{x})\right) \boldsymbol{w}_h^{k+1}(\boldsymbol{x}) \cdot \frac{\partial \boldsymbol{\varphi}_h(\boldsymbol{x})}{\partial x_s} d\boldsymbol{x} \tag{4.18}$$

and

$$\tilde{\sigma}_2 \approx \sigma_2 = \sum_{K_i \in \mathcal{T}_h} \sum_{j \in s(i)} \int_{\Gamma_{ij}} \left[\mathcal{P}^+\left(\left\langle \boldsymbol{w}_h^k \right\rangle_{ij}, \boldsymbol{n}_{ij}\right) \boldsymbol{w}_h^{k+1}|_{\Gamma_{ij}}\right. \tag{4.19}$$

$$\left. + \mathcal{P}^-\left(\left\langle \boldsymbol{w}_h^k \right\rangle_{ij}, \boldsymbol{n}_{ij}\right) \boldsymbol{w}_h^{k+1}|_{\Gamma_{ji}}\right] \cdot \boldsymbol{\varphi}_h dS \tag{4.20}$$

$$+ \sum_{K_i \in \mathcal{T}_h} \sum_{j \in S(i) \setminus s(i)} \int_{\Gamma_{ij}} \boldsymbol{H}\left(\boldsymbol{w}_h^{k+1}|_{\Gamma_{ij}}, \boldsymbol{w}_h^{k+1}|_{\Gamma_{ji}}, \boldsymbol{n}_{ij}\right) \cdot \boldsymbol{\varphi}_h(\boldsymbol{x})\, dS. \tag{4.21}$$

We omit the linearization of the fluxes across domain boundaries, for details see [6]. Finally, we define the form

$$b_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) = -\sigma_1 + \sigma_2. \tag{4.22}$$

The form is linear with respect to the second and third variable. Using this form we come to the following *semi-implicit linearized numerical scheme*: for each $k \geq 0$ find $\boldsymbol{w}_h^{k+1}$ such that

A)

$$\boldsymbol{w}_h^{k+1} \in [V_h]^4 \tag{4.23}$$

B)

$$\left(\boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) + \tau_k b_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) = \left(\boldsymbol{w}_h^k, \boldsymbol{\varphi}_h\right) \quad \forall \boldsymbol{\varphi}_h \in [V_h]^4, \ k = 0, 1, ... \tag{4.24}$$

C)

$$\boldsymbol{w}_h^0 = \Pi_h \boldsymbol{w}^0. \tag{4.25}$$

4.3 Boundary conditions

In all examples in the next chapter, the boundary of the computational domain Ω is divided into three parts.

A) On a fixed impermeable wall $\Gamma_W \subset \partial\Omega$ we use the condition $\boldsymbol{v} \cdot \boldsymbol{n} = 0$. Then the flux $\mathcal{P}(\boldsymbol{w}, \boldsymbol{n})$ has the form

$$\begin{aligned} \mathcal{P}(\boldsymbol{w}, \boldsymbol{n}) &= \sum_{s=1}^{2} \boldsymbol{f}_s(\boldsymbol{w})\, n_s && (4.26)\\ &= (\boldsymbol{v} \cdot \boldsymbol{n})\, \boldsymbol{w} + p\,(0, n_1, n_2, \boldsymbol{v} \cdot \boldsymbol{n})^T && (4.27)\\ &= p\,(0, n_1, n_2, 0)^T, && (4.28) \end{aligned}$$

which is uniquely determined on Γ_W by the extrapolated value of the pressure, i.e. by $p_j^k := p_i^k$. Therefore, on the part Γ_W of the boundary we define the numerical flux

$$\boldsymbol{H}\left(\boldsymbol{w}_i^k, \boldsymbol{w}_j^k, \boldsymbol{n}\right) = p_i^k\,(0, n_1, n_2, 0)^T. \tag{4.29}$$

We can see that on the impermeable part of the boundary, 2 eigenvalues λ_2, λ_3 are zero, the eigenvalue λ_1 is negative, and the eigenvalue λ_4 is positive. We prescribe only $\boldsymbol{v} \cdot \boldsymbol{n} = 0$, and extrapolate the pressure.

B) On the inlet

$$\Gamma_I(t) = \{x \in \partial\Omega; \boldsymbol{v}(x,t) \cdot \boldsymbol{n}(x) < 0\}$$

and the outlet

$$\Gamma_O(t) = \{x \in \partial\Omega; \boldsymbol{v}(x,t) \cdot \boldsymbol{n}(x) > 0\}$$

parts of the boundary it is necessary to use nonreflecting boundary conditions. In this case we used the so-called *characteristics-based* bounadry conditions, according to [10]. Using the rotational invariance, we tranform the Euler equations to the coordinates $\tilde{x}_1$, in the direction of the outer normal $\boldsymbol{n}$ to the boundary, and $\tilde{x}_2$, tangential to the boundary and linearize the resulting system around the state $\boldsymbol{q}_{ij} = \mathbb{Q}(\boldsymbol{n}_{ij})$. Then we obtain the linear system

$$\frac{\partial \boldsymbol{q}}{\partial t} + \mathbb{A}_1\left(\boldsymbol{q}_{ij}\right)\frac{\partial \boldsymbol{q}}{\partial \tilde{x}_1} = 0, \tag{4.30}$$

for the vector-valued function $\boldsymbol{q} = \mathbb{Q}(\boldsymbol{n}_{ij})\,\boldsymbol{w}$, considered in the set $(-\infty, 0) \times (0, \infty)$ and equipped with the initial and boundary conditions

$$\boldsymbol{q}(\tilde{x}_1, 0) = \boldsymbol{q}_{ij}, \quad \tilde{x}_1 < 0, \tag{4.31}$$

$$\boldsymbol{q}(0, t) = \boldsymbol{q}_{ij}, \quad t > 0. \tag{4.32}$$

The goal is to choose $\boldsymbol{q}_{ij}$ in such a way that this initial-boundary value problem is well-posed, i.e. has a unique solutions. The method of characteristics leads to the following process:

Let us put $\boldsymbol{q}^*_{ji} = \mathbb{Q}(\boldsymbol{n}_{ij})\,\boldsymbol{w}^*_{ji}$, where $\boldsymbol{w}_{ji}$ is a prescribed boundary state at the inlet or outlet. We calculate eigenvectors $\boldsymbol{r}_s$ corresponding to the eigenvalues $\lambda_1, ..., \lambda_4$, of the matrix $\mathbb{A}_1\left(\boldsymbol{q}_{ij}\right)$, arrange them as columns in the matrix $\mathbb{T}$ and calculate $\mathbb{T}^{-1}$ (explicit formulae can be found in [9], section 3.1). Now we set

$$\alpha = \mathbb{T}^{-1}\boldsymbol{q}_{ij}, \;\; \beta = \mathbb{T}^{-1}\boldsymbol{q}^*_{ji}. \tag{4.33}$$

and define the state $\boldsymbol{q}_{ji}$ by the relations

$$\boldsymbol{q}_{ji} := \sum_{s=1}^{4} \gamma_s \boldsymbol{r}_s, \;\; \gamma_s = \begin{cases} \alpha_s, & \lambda_s \geq 0, \\ \beta_s, & \lambda_s < 0. \end{cases} \tag{4.34}$$

Finally, the sought boundary state $\boldsymbol{w}_{ji}$ is defined as

$$\boldsymbol{w}|_{\Gamma_{ji}} = \boldsymbol{w}_{ji} = \mathbb{Q}^{-1}(\boldsymbol{n}_{ij})\,\boldsymbol{q}_{ji}. \tag{4.35}$$

4.4 Shock capturing

For the flow containing discontinuities we need to employ suitable stabilization technique to avoid spurious overshoots and undershoots in computed solutions near the discontinuities. This phenomenon does not occur in low Mach number regimes, but in transonic flow it causes instabilities in the numerical solution.

Two approaches will be presented and later used in the numerical examples. The first one, the so-called *vertex based limiter* is taken from [13]. The second one, based on adding *artificial viscosity* into the system of equations is taken from [10].

4.4.1 Vertex based limiter

The vertex based limiter is based on a simple idea of limiting the slope of the solution on an element by limiting the polynomial degree of approximation used on that element. Let v be a component of the numerical solution. Given an element $K_i \in \mathcal{T}_h$, the value of the solution on K_i in the center of gravity of K_i, denoted by v_i^c, and the average value of the gradient of the solution in the center of gravity, denoted by $(\nabla v)_i^c$, the goal is to determine the maximum admissible slope for a constrained reconstruction of the form

$$v_h(\boldsymbol{x}) = v_i^c + \alpha_i (\nabla v)_i^c \cdot (\boldsymbol{x} - \boldsymbol{x}_i^c), \quad 0 \leq \alpha_i \leq 1, \ \boldsymbol{x} \in K_i, \tag{4.36}$$

where $\boldsymbol{x}_i^c$ is the center of gravity of K_i. The correction factor α_i is defined so that the final solution values at the vertices of K_i, denoted in what follows as $\boldsymbol{x}_{i,j}$, $j = 1, ...,$ number of vertices, are bounded by the maximum and minumum values of v in all elements sharing the vertex $\boldsymbol{x}_{i,j}$. We define these values as $v_{i,j}^{\max}, v_{i,j}^{\min}$.

The elementwise correction factors α_i for (4.36) guarantee that

$$v_{i,j}^{\min} \leq v(\boldsymbol{x}_{i,j}) \leq v_{i,j}^{\max}, \ \forall j. \tag{4.37}$$

This vertex-based condition can be enforced using

$$\alpha_i = \min_j \begin{cases} \min\left\{1, \frac{v_{i,j}^{\max} - v_i^c}{v(\boldsymbol{x}_{i,j}) - v_i^c}\right\}, & \text{if } v(\boldsymbol{x}_{i,j}) - v_i^c > 0, \\ 1, & \\ \min\left\{1, \frac{v_{i,j}^{\min} - v_i^c}{v(\boldsymbol{x}_{i,j}) - v_i^c}\right\}, & \text{if } v(\boldsymbol{x}_{i,j}) - v_i^c < 0. \end{cases} \tag{4.38}$$

To limit higher order terms, all derivatives of order p are multiplied by a common correction factor $\alpha_i^{(p)}$. We denote the first component of $(\nabla v)_i^c$ by $\frac{\partial v}{\partial x_1}|_c$, the second by $\frac{\partial v}{\partial x_2}|_c$, analougously for the second derivatives, and by $\overline{f(x_1, x_2)}$ the vol-

ume average of f on K_i. Then we set

$$\begin{aligned} v_h(x_1,x_2) = \overline{v_h} \;&+\; \alpha_i^{(1)} \left\{ \frac{\partial v}{\partial x_1}|_c (x_1 - (x_i^c)_1) + \frac{\partial v}{\partial x_2}|_c (x_2 - (x_i^c)_2) \right\} \\ &+\; \alpha_i^{(2)} \left\{ \frac{\partial^2 v}{\partial x_1^2}|_c \left[\frac{(x_1 - (x_i^c)_1)^2}{2} - \overline{\frac{(x_1 - (x_i^c)_1)^2}{2}} \right] \right. \\ &+\; \frac{\partial^2 v}{\partial x_2^2}|_c \left[\frac{(x_2 - (x_i^c)_2)^2}{2} - \overline{\frac{(x_2 - (x_i^c)_2)^2}{2}} \right] \\ &+\; \left. \frac{\partial^2 v}{\partial x_1 \partial x_2}|_c \left[(x_1 - (x_i^c)_1)(y - (x_i^c)_2) - \overline{(x_1 - (x_i^c)_1)(y - (x_i^c)_2)} \right] \right\}. \end{aligned} \tag{4.39}$$

For details, see [13]. The values of $\alpha_i^{(1)}$ and $\alpha_i^{(2)}$ are determined using the vertex-based limiter, as applied to the linear reconstructions

$$v_{x_1}^{(2)}(x_1,x_2) = \frac{\partial v}{\partial x_1}|_c + \alpha_x^{(2)} \left\{ \frac{\partial^2 v}{\partial x_1^2}|_c (x_1 - (x_i^c)_1) + \frac{\partial^2 v}{\partial x_1 \partial x_2}|_c (x_2 - (x_i^c)_2) \right\}, \tag{4.40}$$

$$v_{x_2}^{(2)}(x_1,x_2) = \frac{\partial v}{\partial x_2}|_c + \alpha_y^{(2)} \left\{ \frac{\partial^2 v}{\partial x_1 \partial x_2}|_c (x_1 - (x_i^c)_1) + \frac{\partial^2 v}{\partial x_2^2}|_c (x_2 - (x_i^c)_2) \right\}, \tag{4.41}$$

$$v^{(1)}(x_1,x_2) = v_h + \alpha_i^{(1)} \left\{ \frac{\partial v}{\partial x_1}|_c (x_1 - (x_i^c)_1) + \frac{\partial v}{\partial x_2}|_c (x_2 - (x_i^c)_2) \right\}. \tag{4.42}$$

The last step is identical to (4.36). In the first and second step, first-order derivatives with respect to x_1 and x_2 are treated in the same way as values in the center of gravity, while second-order derivatives represent the gradients to be limited.

Since the mixed second derivative appears in both (4.40) and (4.41), the correction factor $\alpha_i^{(2)}$ for the limited quadratic reconstruction (4.39) is defined as

$$\alpha_i^{(2)} = \min\left\{ \alpha_{x_1}^{(2)}, \alpha_{x_2}^{(2)} \right\}. \tag{4.43}$$

The general scheme, that is also implemented in the code that was used in calculation of the examples in this thesis, decreases the polynomial degree sequentially in descending order, as opposed to making the assumption that no oscillations are present in v_h if they are not detected in the linear part as proposed in [13]. The outline of the algorithm then is as follows:

The limiting then works for the linear terms in the following way:

4.4.2 Limiter based on adding artificial viscosity

First part of this approach is introducing the discontinuity indicator $g(i)$ proposed in [7] and defined by

$$g(i) = \int_{\partial K_i} \left[\rho_h^k\right]^2 dS / \left(h_{K_i} |K_i|^{3/4} \right), \; K_i \in \mathcal{T}_h. \tag{4.44}$$

Algorithm 2: Limiting algorithm.

```
foreach element do
    p := polynomial degree of the element;
    repeat
        compute α_i^(p);
        if alpha_i^(p) < 1 then
            p = p - 1;
        else
            end the cycle;
    until p = 0;
```

Further we define the discrete indicator

$$G(i) = 0 \text{ if } g(i) < 1 \text{ and } G(i) = 1 \text{ if } g(i) \geq 1, \quad K_i \in \mathcal{T}_h. \tag{4.45}$$

To the left hand-side of (4.24) we add the *artificial viscosity forms*

$$\beta_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}\right) = \nu_1 \sum_{i \in I} h_{K_i} G^k(i) \int_{K_i} \nabla \boldsymbol{w}_h^{k+1} \cdot \nabla \boldsymbol{\varphi} d\boldsymbol{x} \tag{4.46}$$

and

$$J_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}\right) = \nu_2 \sum_{i \in I} \sum_{j \in s(i)} \frac{1}{2}\left(G^k(i) + G^k(j)\right) \int_{\Gamma_{ij}} \left[\boldsymbol{w}_h^{k+1}\right] \cdot [\boldsymbol{\varphi}]\, dS, \tag{4.47}$$

where $\nu_1, \nu_2 \approx 1$. Thus, the resulting scheme reads

A)

$$\boldsymbol{w}_h^{k+1} \in [V_h]^4 \tag{4.48}$$

B)

$$\begin{aligned} \left(\boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) &+ \tau_k b_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}_h\right) + \tau_k \beta_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}\right) \\ &+ \tau_k J_h\left(\boldsymbol{w}_h^k, \boldsymbol{w}_h^{k+1}, \boldsymbol{\varphi}\right) = \left(\boldsymbol{w}_h^k, \boldsymbol{\varphi}_h\right) \quad \forall \boldsymbol{\varphi}_h \in [V_h]^4, \ k = 0, 1, ... \end{aligned} \tag{4.49}$$

C)

$$\boldsymbol{w}_h^0 = \Pi_h \boldsymbol{w}^0. \tag{4.50}$$

An important characteristic of this approach is that $G(i)$ vanishes in regions, where the solution is regular. Therefore, the scheme does not produce any nonphysical entropy in these regions.

Chapter 5

Numerical experiments

In all numerical experiments, we consider the CFL stability condition of the form

$$\tau_k \max_{K_i \in \mathcal{T}_h} \frac{1}{|K_i|} \left(\max_{j \in S(i)} |\Gamma_{ij}| \lambda^{\max}_{\mathcal{P}\left(\boldsymbol{w}_h^k|_{\Gamma_{ij}}, \boldsymbol{n}_{ij}\right)} \right) \leq CFL. \tag{5.1}$$

Here, CFL is a given constant and $\lambda^{\max}_{\mathcal{P}\left(\boldsymbol{w}_h^k|_{\Gamma_{ij}}, \boldsymbol{n}_{ij}\right)}$ is the maximum over Γ_{ij} of the spectral radius of the matrix $\mathcal{P}\left(\boldsymbol{w}_h^k|_{\Gamma_{ij}}, \boldsymbol{n}_{ij}\right)$.

Also in all numerical experiments, we consider the medium in which we study the flow to be air, and therefore we set $\kappa = 1.4$.

In the light of 4.3, the prescribed boundary states are derived from primitive variables denoted as $\rho_{EXT}, v_{1_{EXT}}, v_{2_{EXT}}, p_{EXT}$ using the transformations described in Subsection 1.2.2.

5.1 Transonic flow

The first three examples demonstrate transonic flows with high Mach numbers, where applying of shock capturing is crucial.

5.1.1 Reflected shock benchmark

This is a classic benchmark for the solution of compressible flow described in [23]. The prescribed relative tolerance TOL for guiding the *hp*-adaptivity was set to 0.5%. There are two different prescribed boundary states, corresponding to the left, and to the top part of the boundary, and are derived from the following

prescribed values.

$$\rho_{EXT}^{LEFT} = 1.0 \tag{5.2}$$
$$v_{1_{EXT}^{LEFT}} = 2.9 \tag{5.3}$$
$$v_{2_{EXT}^{LEFT}} = 0.0 \tag{5.4}$$
$$p_{EXT}^{LEFT} = 0.714286 \tag{5.5}$$
$$\tag{5.6}$$
$$\rho_{EXT}^{TOP} = 1.7 \tag{5.7}$$
$$v_{1_{EXT}^{TOP}} = 2.619334 \tag{5.8}$$
$$v_{2_{EXT}^{TOP}} = -0.5063. \tag{5.9}$$
$$p_{EXT}^{TOP} = 1.52819 \tag{5.10}$$

The initial state is chosen to be the one prescribed on the left part of the boundary. In the solution of this problem, we employed the shock capturing scheme described in 4.4.1.

In Figures 5.1 - 5.10, the solution and corresponding meshes (the whole mesh and a detail) at various time levels is shown.

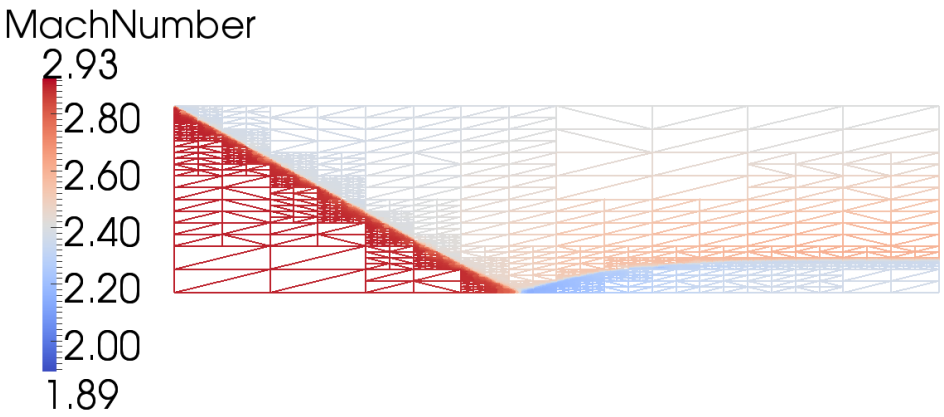

Figure 5.1: Solution, time = 0.879538.

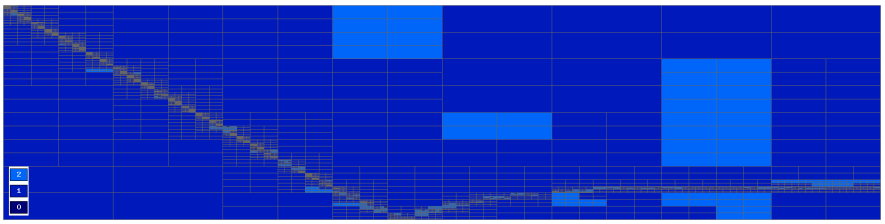

Figure 5.2: Mesh and polynomial orders, time = 0.879538, number of DOFs: 80672.

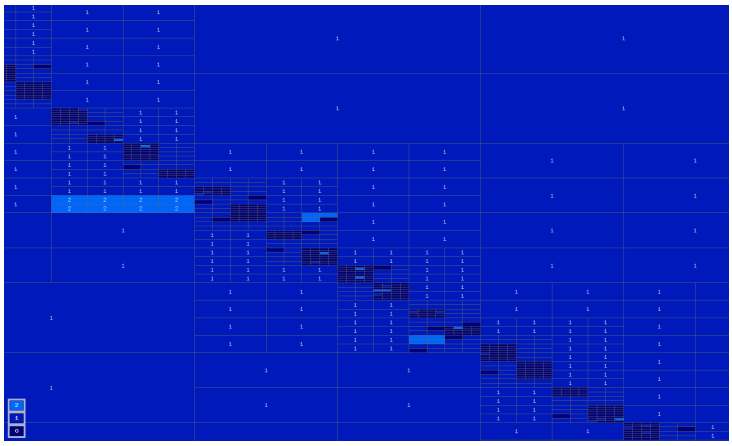

Figure 5.3: Detail of the mesh, layer on the left, time = 0.879538.

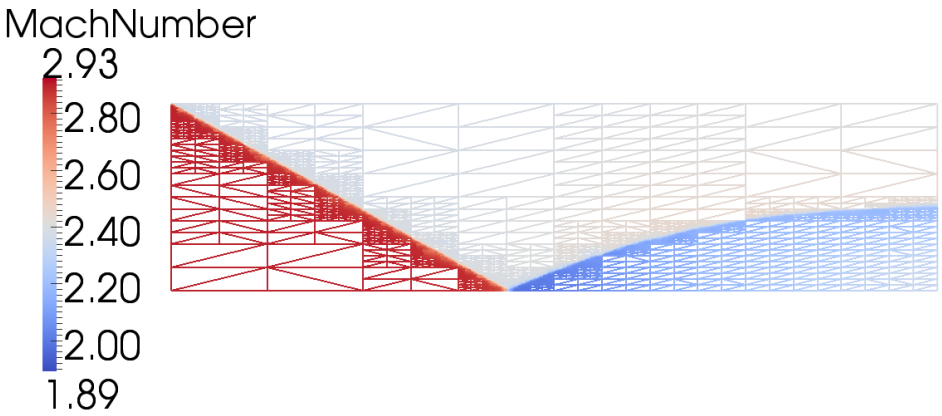

Figure 5.4: Solution, time = 1.4307.

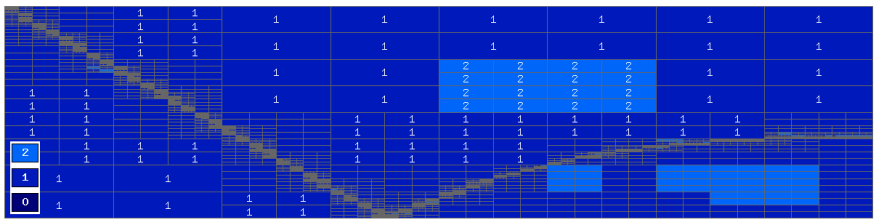

Figure 5.5: Mesh and polynomial orders, time = 1.4307, number of DOFs: 109824.

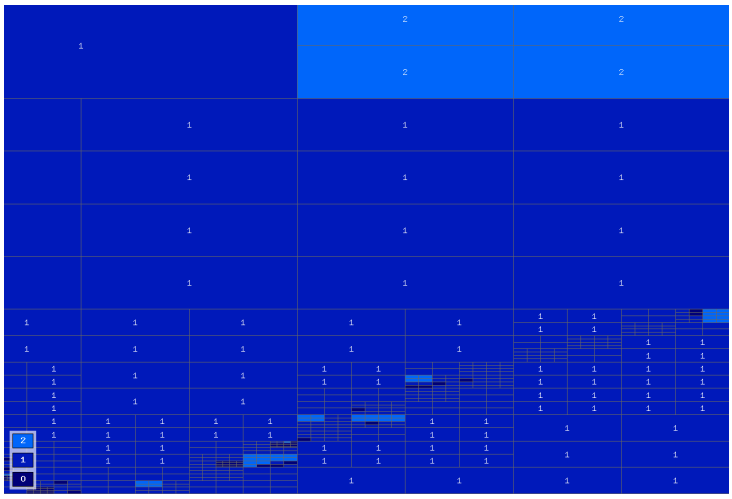

Figure 5.6: Detail of the mesh, middle bottom, time = 1.4307.

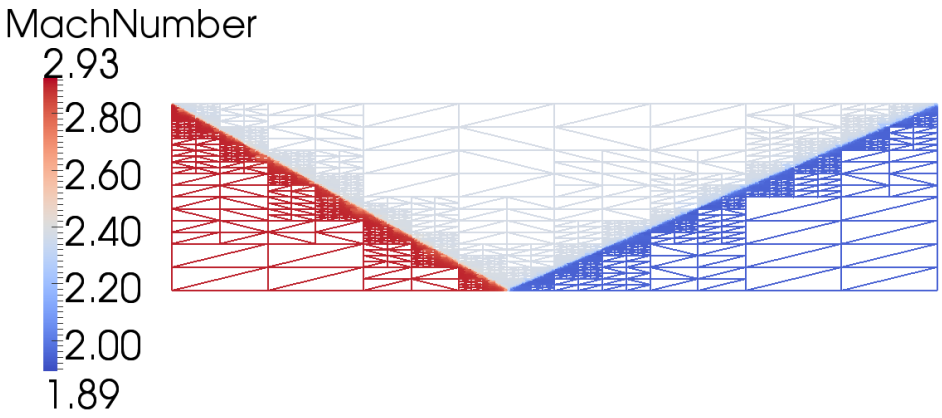

Figure 5.7: Final solution.

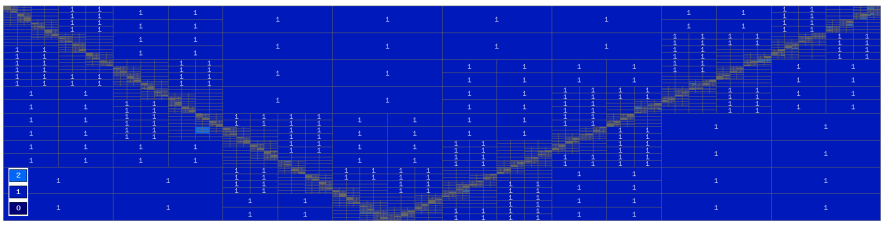

Figure 5.8: Final mesh and polynomial orders, number of DOFs: 120736.

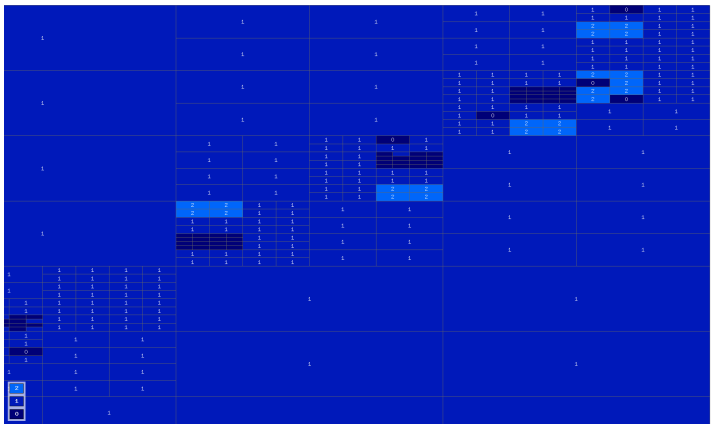

Figure 5.9: Detail of the final mesh, top-right corner.

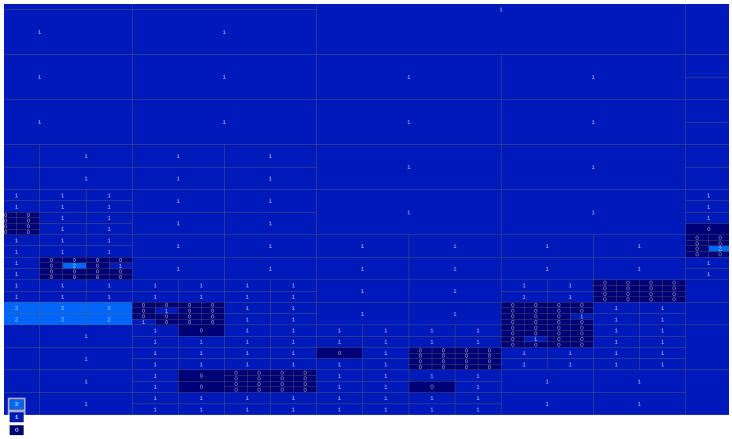

Figure 5.10: Detail of the final mesh, middle bottom.

In Figure 5.11, the same comparison to the exact solution, as in [23] was carried out, with more satysfying results.

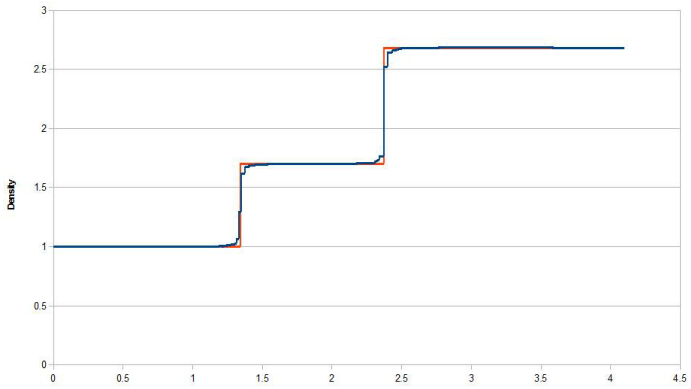

Figure 5.11: Density along y = 0.25, exact solution(red), and calculated solution(blue). See [23], Fig. 4, for comparison

5.1.2 GAMM channel

This is another classic benchmark for the solution of compressible flow.

Domain The domain is a horizontal channel with a 10% circular arc bump on the lower wall as described in [9], Section 3.7.2. With respect to Section 4.3, we denote

$$\begin{aligned}
\text{inlet: } \Gamma_I &= \left\{\boldsymbol{x} \in \mathbb{R}^2; x_1 = -1, x_2 \in (0,1)\right\}, && (5.11)\\
\text{outlet: } \Gamma_O &= \left\{\boldsymbol{x} \in \mathbb{R}^2; x_1 = 1, x_2 \in (0,1)\right\}, && (5.12)\\
\text{lower wall: } \Gamma_W^1 &= \left\{\boldsymbol{x} \in \mathbb{R}^2; x_1 \in [-1,-0.5], x_2 = 0\right\} && (5.13)\\
&\cup \left\{\boldsymbol{x} \in \mathbb{R}^2; x_1 \in [-0.5,0.5], x_2 = \sqrt{1.69 - x_1^2} - 1.2\right\} && (5.14)\\
&\cup \left\{\boldsymbol{x} \in \mathbb{R}^2; x_1 \in [0.5,1], x_2 = 0\right\}, && (5.15)\\
\text{outlet: } \Gamma_W^2 &= \left\{\boldsymbol{x} \in \mathbb{R}^2; x_1 \in [-1,1], x_2 = 1\right\}. && (5.16)
\end{aligned}$$

The prescribed primitive variables for the boundary state are as follows:

$$\begin{aligned}
\rho_{EXT} &= 1.0 && (5.17)\\
v_{1_{EXT}} &= 1.25 && (5.18)\\
v_{2_{EXT}} &= 0.0. && (5.19)\\
p_{EXT} &= 2.5 && (5.20)
\end{aligned}$$

In the solution of this problem, we employed the shock capturing scheme described in 4.4.2 which proved to be much better in terms of resolution than the vertex-based limiter described in 4.4.1. The prescribed relative tolerance TOL for guiding

the *hp*-adaptivity was set to 0.45%, later decreased to 0.25% for better resolution of the shock wave. In Figures 5.12 - 5.17, the solution and corresponding mesh at various time levels is shown. The distribution of Mach number, and density on the lower wall are shown in Figures 5.19, and 5.20 respectively. Detail of the final mesh is shown in Figure 5.18.

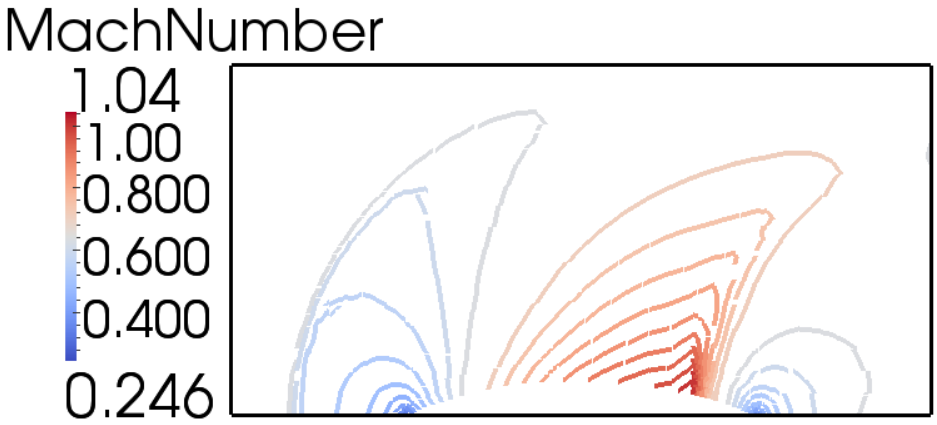

Figure 5.12: Solution, time = 0.469071.

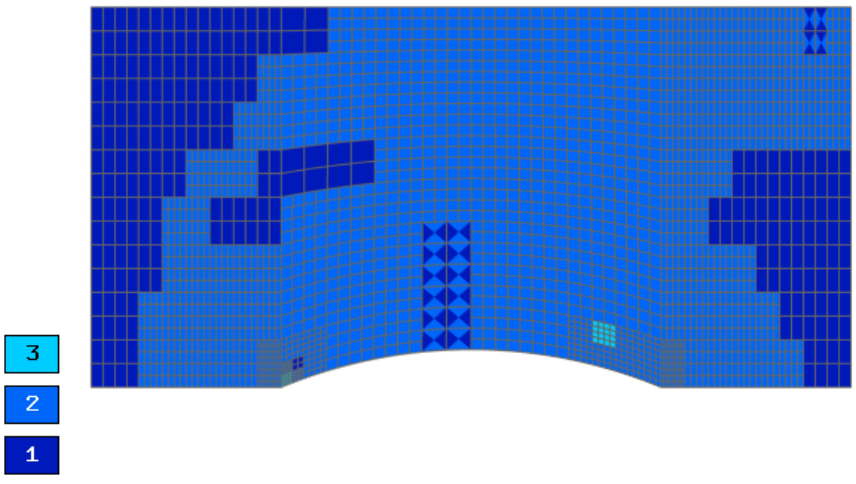

Figure 5.13: Mesh and polynomial orders, time = 0.469071, number of DOFs: 62816.

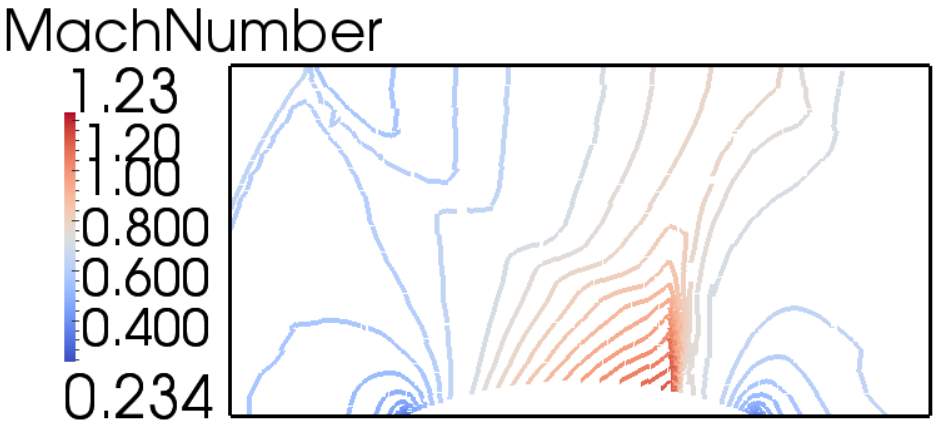

Figure 5.14: Solution, time = 0.900324.

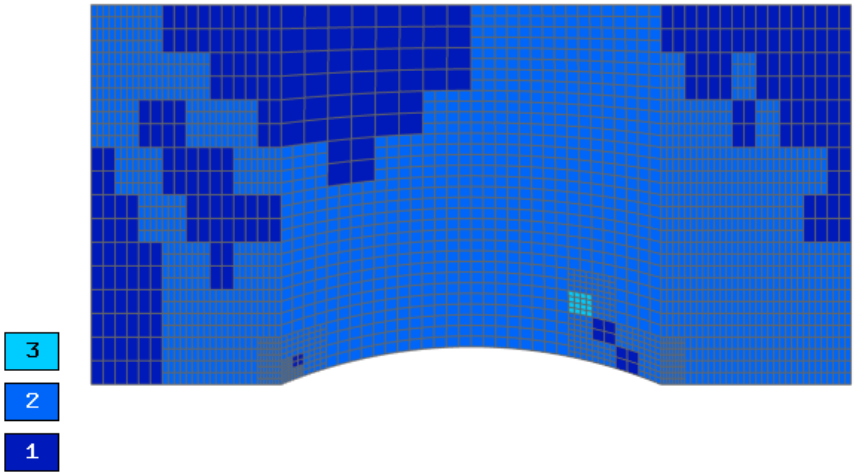

Figure 5.15: Mesh and polynomial orders, time = 0.900324, number of DOFs: 75904.

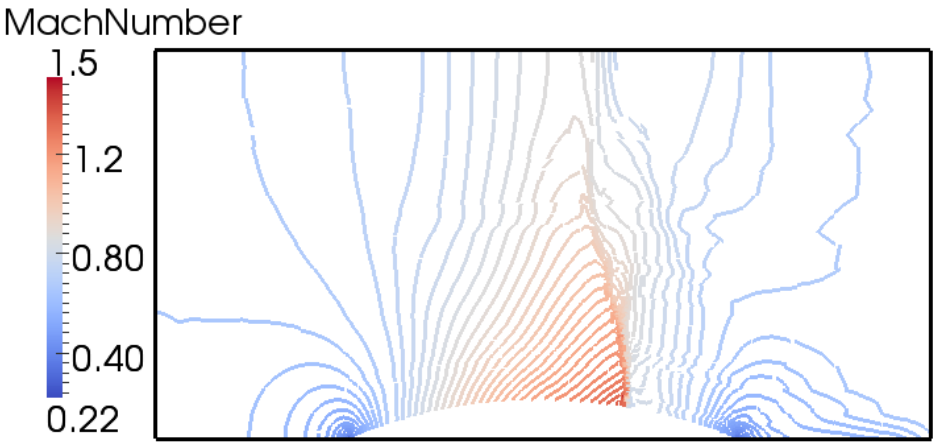

Figure 5.16: Solution, time = 4.46887.

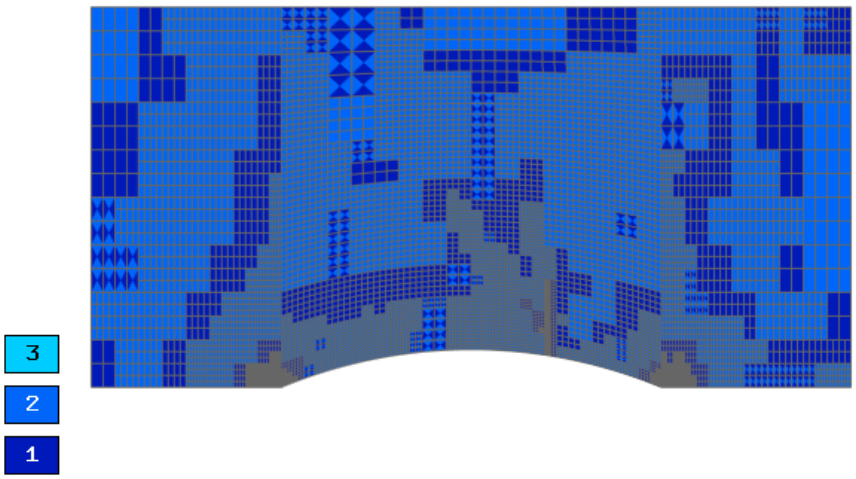

Figure 5.17: Mesh and polynomial orders, time = 4.46887, number of DOFs: 302880.

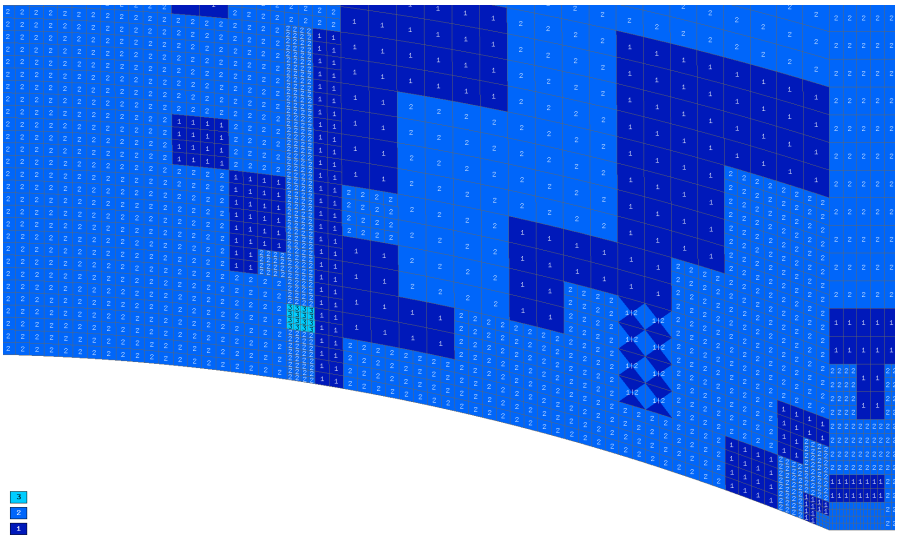

Figure 5.18: Detail of the final mesh.

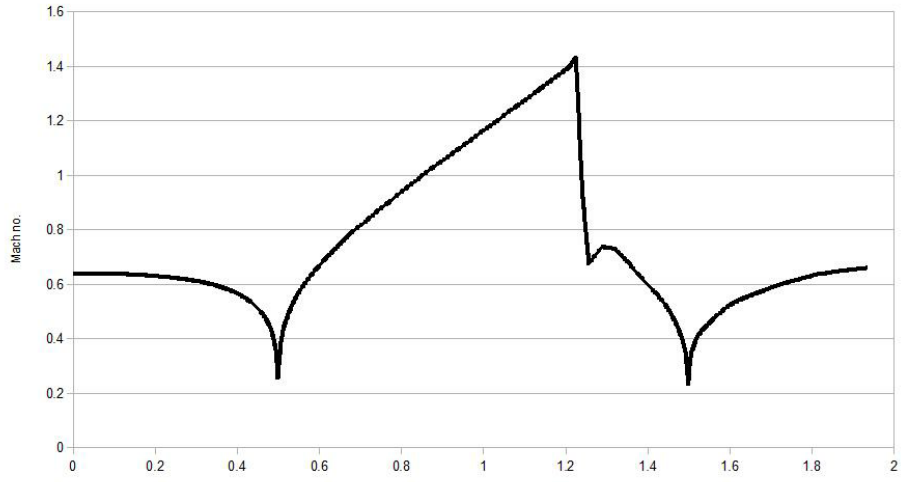

Figure 5.19: Final distribution of Mach number along the lower wall.

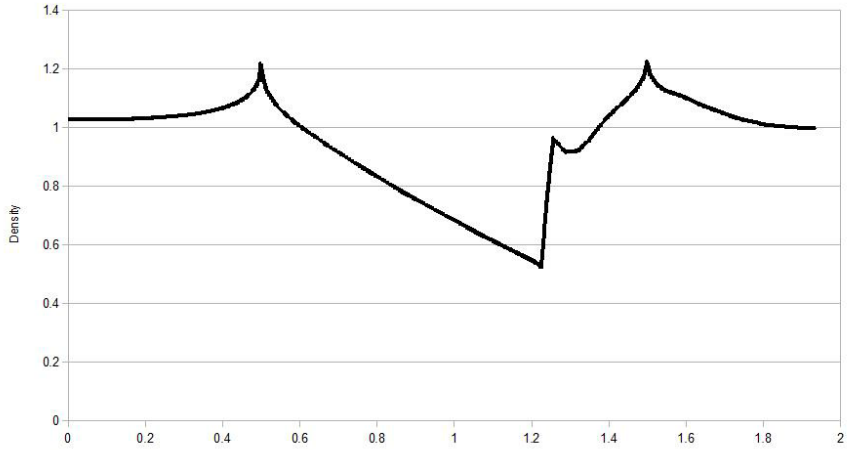

Figure 5.20: Final distribution of density along the lower wall.

5.1.3 Forward facing step

This is a classic benchmark for the solution of compressible flow. In the light of 4.3, the prescribed boundary state is derived from the following prescribed values.

$$\rho_{EXT} = 1.4 \tag{5.21}$$
$$v_{1_{EXT}} = 3.0 \tag{5.22}$$
$$v_{2_{EXT}} = 0.0 \tag{5.23}$$
$$p_{EXT} = 1.0 \tag{5.24}$$

To be completed.

Conclusion

In the first chapters, a brief introduction to the problem we solve (flow of compressible media) was given, together with derivation of equations describing the flow. Some properties of the system of equations we then solved, important for creation of the numerical schemes, and for the solution, were presented. The purpose of the very short comparison of various methods (Finite Element method, Finite Volume method) possible to be employed to solve problems of flow was to emphasize the qualities of the DG method. Description of the adaptive strategies used for the numerical solution was given and some aspects were emphasized, or even shown in computer code in appendix. The application of the DG method on the system of equations at hand was carried out with detail, and the DG method was successfully applied to the standard problems in the solution of the compressible flow.

The *hp*-adaptive algorithm gave very good resolution of the quantity distributions that we were interested in, it also kept our linear systems sufficiently small so that the calculations were possible to be carried out on a normal PC. From the computational results we could see though that the algorithm pays for its universality and robustness by longer time of the computation.

Outlook

A) The semi-implicit scheme suffers from very short time-steps that only satisfy the CFL condition, fully implicit schemes would not have this disadvantage, although their derivation is usually much more difficult

B) We have to be aware that more challenges for *hp*-adaptivity algorithms appear in 3-dimensional setting (more ways to refine an element, more integration points for higher order elements), but interesting applications usually require the 3-dimensional setting

C) The algorithm can still be optimized in terms of parallelization, especially the possibility to run it on a cluster of many processors, or over the internet.

Appendix

Appendix 1 - Source code snippets

Details of the data structures We are maintaining two independent layers of nodes: the first layer contains all vertex nodes, the second all edge nodes. The following two functions can be called:

```
Node* get_vertex_node(int id1, int id2);
Node* get_edge_node(int id1, int id2);
```

All nodes, apart from being accessible by their `id` number, can be reached using these functions by passing the `id`s of their "parent" nodes. Top-level vertex nodes (those loaded from the mesh file) are stored at the beginning of the node array and can be accessed directly without hashing. Mesh initialization then becomes trivial:

```
for all Elements
  vertex_numbers = read element vertex id numbers
  for all edges of the Element
    Element.vn = vertex_numbers
    Element.en[0] = get_edge_node(vertex_numbers[0], vertex_numbers[1]);
    Element.en[1] = get_edge_node(vertex_numbers[1], vertex_numbers[2]);
    Element.en[2] = get_edge_node(vertex_numbers[2], vertex_numbers[3]);
    // In case of triangles
    Element.en[3] = get_edge_node(vertex_numbers[3], vertex_numbers[0]);
```

Element refinement is also very straightforward. No care must be taken of the neighboring elements, regardless of their refinement level or even existence:

```
create_triangle(Node v0, Node v1, Node v2)
{
  Element e;
  e.vn[0] = v0;
  e.vn[1] = v1;
  e.vn[2] = v2;
  e.en[0] = get_edge_node(v0.id, v1.id);
  e.en[1] = get_edge_node(v1.id, v2.id);
  e.en[2] = get_edge_node(v2.id, v0.id);
}

void refine_triangle(Element e)
{
  Node x0 = get_vertex_node(e.vn[0].id, e.vn[1].id);
  Node x1 = get_vertex_node(e.vn[1].id, e.vn[2].id);
  Node x2 = get_vertex_node(e.vn[2].id, e.vn[0].id);
  e.sons[0] = create_triangle(e.vn[0], x0, x2);
  e.sons[1] = create_triangle(x0, e.vn[1], x1);
  e.sons[2] = create_triangle(x2, x1, e.vn[2]);
  e.sons[3] = create_triangle(x0, x1, x2);
  e.active = 0;
}

create_quad(Node v0, Node v1, Node v2)
{
  Element e;
  e.vn[0] = v0;
  e.vn[1] = v1;
  e.vn[2] = v2;
```

```
  e.vn[3] = v3;

  e.en[0] = get_edge_node(v0.id, v1.id);
  e.en[1] = get_edge_node(v1.id, v2.id);
  e.en[2] = get_edge_node(v2.id, v3.id);
  e.en[3] = get_edge_node(v3.id, v0.id);
}

void refine_quad(Element e)
{
  Node x0 = get_vertex_node(e.vn[0].id, e.vn[1].id);
  Node x1 = get_vertex_node(e.vn[1].id, e.vn[2].id);
  Node x2 = get_vertex_node(e.vn[2].id, e.vn[3].id);
  Node x3 = get_vertex_node(e.vn[3].id, e.vn[0].id);
  e.sons[0] = create_quad(e.vn[0], x0, mid, x3);
  e.sons[1] = create_quad(x0, e.vn[1], x1, mid);
  e.sons[2] = create_quad(mid, x1, e.vn[2], x2);
  e.sons[3] = create_quad(x3, mid, x2, e.vn[3]);
  e.active = 0;
}
```

Each hash table is implemented as an array of linked lists. This hash table organization has the advantage of simple node removal, which is required if a node is no longer needed by any element.

Finding all neighbors of an element - algorithm

```
void find_neighbors(Element* element, int edge)
{
// Try to get a neighbor directly from the structure (as explained above)
 neighb_el = element->get_neighbor(edge);
 // If successful, this is the easy case.
 if (neighb_el != NULL)
 {
  // There is only one neighbor in this case.
  n_neighbors = 1;
  neighbors.push_back(neighb_el);
 }
 else
 {
  // Peek the vertex in the middle of the active edge
  // (if there is none, vertex will be NULL).
  Node* vertex = mesh->peek_vertex_node(element->en[edge]->p1,
  element->en[edge]->p2);
  // There is no vertex in the middle of the active edge,
  // from the point of view of this element.
  // We call this case the "way up".
  if (vertex == NULL)
  {
   // Get the parent element.
   Element* parent = central_el->parent;
   // Array of middle-point vertices of the intermediate parent
   // edges that we climb up to the correct parent element.
   Node** par_mid_vertices = new Node*[max_level];
   // Number of visited intermediate parents.
   int n_parents = 0;
   // Function will be displayed later, it looks for an active element going up.
   find_act_elem_up(parent, orig_vertex_id, par_mid_vertices, n_parents);
  }
  // There is a vertex in the middle of the current element's current edge.
  // We call this the "way down".
```

```
  else
  {
   // An array of virtual sons of the element visited on the way down
   // to the neighbor.
   int sons[Transformations::max_level];
   // Number of used transformations.
   int n_sons = 0;
   // Start the search by going down to the first son.
   // Function will be displayed later.
   find_act_elem_down( vertex, orig_vertex_id, sons, n_sons + 1);
  }
 }
}

void find_act_elem_up(Element* elem, int* orig_vertex_id,
Node** par_mid_vertices, int n_parents)
{
 // IDs of vertices bounding the current intermediate parent edge.
 int p1 = elem->vn[edge]->id;
 int p2 = elem->vn[(edge + 1) \% elem->get_num_surf()]->id;

 // Find if p1 and p2 bound a used edge (used by the neighbor element).
 common_edge = mesh->peek_edge_node(p1, p2);

 // Add the vertex in the middle of the parent edge to the array
 // of intermediate parent vertices. This is for consequent transformation
 // of functions on neighbor element.
 vertex = mesh->peek_vertex_node(p1, p2);
 if(vertex != NULL)
 {
  if (n_parents == 0)
   par_mid_vertices[n_parents++] = vertex;
  else
   if(par_mid_vertices[n_parents - 1]->id != vertex->id)
    par_mid_vertices[n_parents++] = vertex;
 }

 if ((common_edge == NULL) || (central_el->en[edge]->id == common_edge->id))
 {
  // We have not yet found the parent of the central element completely
  // adjacent to the neighbor.
  find_act_elem_up(elem->parent, orig_vertex_id, par_mid_vertices, n_parents);
 }
 else
 {
  for (int i = 0; i < 2; i++)
  {
   // Get a pointer to the active neighbor element.
   if ((edge->elem[i] != NULL) && (edge->elem[i]->active == 1))
   {
    neighb_el = edge->elem[i];
    Node* n = NULL;

    // Go back through the intermediate inactive parents down to the central
    // element and stack corresponding neighbor_transformations into the array
    // 'neighbor_transformations'.
    for(int j = n_parents - 1; j > 0; j--)
    {
     n = mesh->peek_vertex_node(par_mid_vertices[j]->id, p1);
     if(n == NULL)
      p1 = par_mid_vertices[j]->id;
     else
     {
      if(n->id == par_mid_vertices[j-1]->id)
```

```
        p2 = par_mid_vertices[j]->id;
       else
        p1 = par_mid_vertices[j]->id;
      }
     }
     // There is only one neighbor, ...
     n_neighbors = 1;
     // ...add it to the vector of neighbors.
     neighbors.push_back(neighb_el);
    }
   }
  }
}

find_act_elem_down( Node* vertex, int* bounding_verts_id,
int* sons, unsigned int n_sons)
{
 // We are looking for neighboring elements on both halves of the edge.
 for (int i = 0; i < 2; i++)
 {
  // Store the current element's transformation.
  sons[n_sons-1] = (edge + i) \% central_el->get_num_surf();
  // Try to get a pointer to the edge between the middle vertex and
  // one of the vertices bounding the previously tested segment.
  Node* common_edge = mesh->peek_edge_node(mid_vert, bnd_verts[i]);
  // If the edge is not used, i.e. there is no active element on either side.
  if (common_edge == NULL)
  {
   // Get the middle vertex of this edge and try again on the segments
   // into which this vertex splits the edge.
   Node * n = mesh->peek_vertex_node(mid_vert, bnd_verts[i]);
   // Choose the correct bounding vertices for the current half of the edge.
   if(i == 0)
    bounding_verts_id[1] = mid_vert;
   else
    bounding_verts_id[0] = mid_vert;
   // Recursively call the function.
   find_act_elem_down( n, bounding_verts_id, sons, n_sons + 1);
   // Fix the recursion.
   bounding_verts_id[0] = bnd_verts[0];
   bounding_verts_id[1] = bnd_verts[1];
  }
  // We have found a used edge.
  else
  {
   // Try both halves and see if the neighboring elements are active
   // (not refined further).
   for (int j = 0; j < 2; j++)
   {
    if ((edge->elem[j] != NULL) && (edge->elem[j]->active == 1))
    {
     neighb_el = mesh->get_element(common_edge->elem[j]->id);
     // Append the new neighbor.
     n_neighbors++;
     neighbors.push_back(neighb_el);
    }
   }
  }
 }
}
```

Bibliography

[1] Aksoylu B., Bond S., Holst M.: An Odyssey into Local Refinement and Multilevel Preconditioning III: Implementation and Numerical Experiments, SIAM J. Sci. Comput., Vol. 25, pp. 478 - 498, 2003.

[2] Babuška I., Práger M., Vitásek E.: *Numerical Processes in Differential Equations*, STNL Praha - Wiley, 1966.

[3] Bornemann, F.: An adaptive multilevel approach to parabolic equations I. General theory and 1D-implementation, IMPACT of Comput. Sci. Engrg. 2, 279 - 317, 1990

[4] Demkowicz L., Oden J.T., Rachowicz W., Hardy O.: Toward a universal *hp*-adaptive finite element strategy. Part 1: constrained approximation and data structure. Comput. Methods Appl. Math. Engrg., **77**, 79 - 112, 1989.

[5] Demkowicz L., Rachowicz W., Devloo P.: A Fully Automatic *hp*-Adaptivity. TICAM Report No. 01-28, University of Texas at Austin, 2001.

[6] Dolejší V., Feistauer M.: *A semi-implicit discontinuous Gelerkin finite element method for the numerical solution of inviscid compressible flow*, J. Comput. Phys., **198**, 727 - 746, 2004.

[7] Dolejší V., Feistauer M., Schwab C.: *On some aspects of the discontinuous Galerkin finite element method for conservation laws*, Math. Comput. Simul., **6**, 333 - 346, 2003.

[8] Feistauer M.: *Mathematical methods in fluid dynamics*, Longman Scientific & Technical, 1993.

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

[10] Feistauer M., Kučera V.: *A New Technique for the Numerical Solution of the Compressible Euler Equations with Arbitrary Mach Numbers*, Proc. of the Conference Hyperbolic Problems:Theory, Numerics and Applications, Springer-Verlag Berlin-Heidelberg, 523 - 531, 2008.

[11] Girault V., Raviart P. - A.:*Finite Element Methods for Navier-Stokes Equations*, Springer, Berlin, 1986.

[12] John, V., Knobloch, P.: *On discontinuity-capturing methods for convection-diffusion equations*, Numerical Mathematics and Advanced Applications, Proceedings of ENUMATH 2005, Springer-Verlag, Berlin, 336 - 344, 2006.

[13] D. Kuzmin.: *A vertex-based hierarchical slope limiter for p-adaptive discontinuous Galerkin methods*, J. Comput. Appl. Math., **233(12)**, 3077-3085, 2010.

[14] Karniadakis G.E., Sherwin S.J.: *Spectral/hp Element Methods for CFD*, Oxford University Press, Oxford, 1999.

[15] Lube G.:*Stabilized Galerkin finite element methods for convection dominated and incompressible flow problems*, Num. Anal. and Math. Model. **29**, Banach Center publications, Warszawa, 1994.

[16] Nitsche, J.A.: *Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind.*, Abh. Math. Sem. Univ. Hamburg, **36**, 9 - 15, 1970/1971.

[17] Paszynski M., Kurtz J., Demkowicz L.: *Parallel, fully automatic hp-adaptive 2D finite element package*, TICAM Report 04-07, The University of Texas at Austin, 2004.

[18] Rachowicz W., Demkowicz L.: *An hp-adaptive finite element method for electromagnetics. Part II. A 3D implementation*, Internat. J. Numer. Methods Engrg., **53**, 147 - 180, 2002.

[19] Šolín P.: Partial Differential Equations and the Finite Element Method, J. Wiley & Sons, New Jersey, 2006

[20] Šolín P., Demkowicz L.: Goal-Oriented hp-Adaptivity for Elliptic Problems, Comput. Methods Appl. Mech. Engrg., **193**, 449 - 468, 2004.

[21] Šolín P., Segeth K. and Doležel I.: *Higher-Order Finite Element Methods*, Chapman & Hall/CRC Press, Boca Raton, 2003.

[22] Šolín P., Vejchodský T., Zítka M.: Orthogonal hp-FEM for Elliptic Problems Based on a Non-Affine Concept. In: Proceedings of ENUMATH 2005, Springer-Verlag, 2005

[23] Tayfun E. Tezduyar, Masayoshi Senga: Stabilization and shock-capturing parameters in SUPG formulation of compressible flows, Comput. Methods Appl. Mech. Engrg. 195, 1621 – 1632, 2006

[24] Zlámal M.: *On the finite element method*, Numer. Math. 12, 394 – 409, 1968.