Technische Universität Dresden

Fakultät Maschinenwesen
Institut für Luft- und Raumfahrttechnik

Diplomarbeit

ILR–TFD D 06–27

Anisotropic Mesh Refinement for Discontinuous Galerkin Methods in Aerodynamic Flow Simulations

Tobias Leicht

eingereicht am 2. November 2006

Betreuer: Prof. Dr.-Ing. Roger Grundmann (1. Gutachter)
Dr. Ralf Hartmann, DLR/TU Braunschweig (2. Gutachter)
Dr.-Ing. Jörg Stiller

Contents

1. Introduction **1**

2. Governing Equations of Computational Fluid Dynamics **3**
2.1. Hyperbolic conservation equations . . . 3
2.2. Euler equations . . . 4
2.3. Navier–Stokes equations . . . 5

3. Discontinuous Galerkin methods **7**
3.1. Discontinuous function spaces . . . 7
3.2. Discretization and notation . . . 8
3.3. Discretization of the compressible Euler equations . . . 9
3.4. Discretization of the compressible Navier–Stokes equations . . . 11
3.5. Boundary conditions . . . 12

4. Mesh refinement **14**
4.1. Mesh refinement techniques . . . 14
4.1.1. h–refinement . . . 14
4.1.2. p–refinement . . . 15
4.1.3. hp–refinement . . . 15
4.1.4. Node movement, r–refinement . . . 15
4.1.5. Remeshing . . . 16
4.2. Adaptive mesh refinement . . . 16

5. Error estimation **19**
5.1. Hierarchical error estimation . . . 19
5.2. Adjoint based indicators . . . 20
5.3. Residual based indicators . . . 23

6. Anisotropic refinement **25**
6.1. Definition of isotropic and anisotropic refinement . . . 25
6.2. Anisotropic refinement indicators . . . 28
6.2.1. Jump indicator . . . 29
6.2.2. Derivative indicators . . . 30
6.2.3. Other sources of information on anisotropic features . . . 36

7. Numerical results **38**
7.1. Expected convergence behavior . . . 38
7.2. Test cases . . . 39
7.2.1. NACA0012 . . . 39
7.2.2. Test case A: Viscous subsonic flow . . . 40

7.2.3. Test case B: Inviscid transonic flow 40
7.2.4. Test case C: Viscous supersonic flow 41
7.3. Results for the jump indicator 41
7.4. Results for derivative indicators 45
7.5. Comparison and indicator recommendation 48
7.5.1. Comparison for test case A 48
7.5.2. Comparison for test case B 49
7.5.3. Comparison for test case C 52
7.5.4. Indicator recommendation 55

8. Conclusion and outlook **56**

Bibliography **59**

A. Implementational issues **61**
A.1. Status of `deal.II` and data storage structure 61
A.2. Connectivity 62
A.3. Prolongation and restriction operators 65
A.4. Mesh smoothing 67

B. Metric intersection **70**

List of Figures

3.1. Mapping σ_κ of reference element $\hat{\kappa}$ to the element κ in real space. 9
3.2. Definition of the interior and outer traces $\mathbf{v}_\kappa^\pm$ wrt. element κ. 9

6.1. Isotropic refinement for quadrilateral elements. 25
6.2. Isotropic refinement for the hexahedral reference element. 26
6.3. Anisotropic refinement for quadrilateral elements, refinement cases `cut_x` and `cut_y`. 27
6.4. Anisotropic refinement cases for the hexahedral reference element. 28

7.1. Initial mesh around a NACA0012 airfoil of unit length, 3072 elements. . . 40
7.2. Test case A: viscous flow at $M = 0.5$, $\alpha = 0°$ and $Re = 5000$: Mach isolines (top) and streamlines (bottom). 41
7.3. Test case B: inviscid flow at $M = 0.8$ and $\alpha = 1.25°$: Mach isolines (top) and streamlines (bottom). 42
7.4. Test case C: viscous flow at $M = 1.2$, $\alpha = 0°$ and $Re = 1000$: Mach isolines (top), and streamlines (bottom). 42
7.5. Test case A: Convergence of the error in c_{dp} for the DG(1) method using adjoint based error indicators combined with anisotropic jump indicators. 43
7.6. Test case A: Convergence of the error in c_{dp} for the DG(2) method using adjoint based error indicators combined with an anisotropic jump indicator. 44
7.7. Test case A: Convergence of the error in c_{dp} for the DG(1) method using residual and adjoint based error indicators combined with an anisotropic jump indicator. 44
7.8. Test case A: Convergence of the error in c_{dp} for the DG(1) method using adjoint based error indicators combined with anisotropic derivative indicators. 45
7.9. Combinations of threshold ratio θ_r and threshold angle θ_α resulting in the modified threshold ratio $\tilde{\theta}_r = 2.53$. 46
7.10. Test case A: Convergence of the error in c_{dp} for the DG(2) method using adjoint based error indicators combined with an anisotropic derivative indicator. 47
7.11. Test case A: Convergence of the error in c_{dp} for the DG(1) method using residual and adjoint based error indicators combined with an anisotropic derivative indicator. 48
7.12. Test case A: Convergence of the error in c_{dp} for the DG(1) method using adjoint based error indicators combined with anisotropic jump and derivative indicators. 48
7.13. Test case A: Convergence of the error in c_{dp} for the DG(2) method using adjoint based error indicators combined with anisotropic jump and derivative indicators. 49

7.14. Test case A: Adapted mesh after three refinement steps using adjoint based error indicators for the DG(2) method. 50
7.15. Test case B: Convergence of the error in c_{dp} for the DG(1) method using residual and adjoint based error indicators combined with anisotropic jump and derivative indicators. 51
7.16. Test case B: Adapted anisotropic meshes after six refinement steps for the DG(1) method using a derivative indicator based on the Mach number. . . 52
7.17. Test case C: Adapted anisotropic meshes after five refinement steps for the DG(1) method using a derivative indicator based on the conservative variables: distant view (left) and zoom of the trailing edge (right). 53
7.18. Test case C: Convergence of the error in c_{df} for the DG(1) method using residual and adjoint based error indicators combined with anisotropic jump and derivative indicators. 54

A.1. Example of a refinement case with a line (edge, face) belonging to elements of more than one level. 62
A.2. Example of a refinement case resulting in corrupted data. 62
A.3. Example of a refinement case with the coarser element being on a higher level. 63
A.4. Example of a refinement case with a refined neighbor, which is not the stored neighbor's (direct) child. 64

B.1. Approximate intersection of two ellipses representing two metrics. 72

1. Introduction

A body immersed in a moving fluid is exposed to forces and torques resulting from the flow. For some applications these loads have to be considered as external perturbations, important examples are bridges or high buildings. In other applications, however, the loads resulting from the flow field are an important part of the functionality of an object, consider for example turbines or aircraft.

In both cases, a good knowledge of the flow field and the resulting forces and torques is required to ensure the functionality of the technical objects under consideration. This knowledge cannot be acquired by practical tests and measurements alone, as such experiments are expensive, require a lot of preparation time and often include scaled models, which cannot always give appropriate results. Therefore, it is important to gain knowledge of flow features without experiments. One way to achieve this is by using numerical methods to solve the equations describing flow phenomena. These governing equations are given in Chapter 2 for the case of aerodynamic applications, i.e. compressible gas flows.

Numerical methods can help saving considerable cost and time, especially in the development process. However, numerical methods need resources of their own, mainly a combination of computing hardware and the time needed for solving the problem with this hardware[1]. It is, therefore, desirable to develop numerical methods, which do not only lead to detailed and precise approximations of the flow but do so with minimized effort.

One way of achieving efficient calculations is to use methods with a high order of convergence. Among these, Discontinuous Galerkin methods (DG methods for short) have become very popular recently, as they offer some considerable advantages over classical Finite Volume methods, for example. The general idea of these methods will be described in Chapter 3. Along with that, the discretization of the governing equations will be given. The concept of adaptivity offers an additional approach of gaining high efficiency. The flow problem is solved using an initial discretization which in general is too coarse. However, after the solution is obtained, the discretization is refined in those regions, where large errors are produced. The obtained discrete problem is solved, starting a new iteration of the adaptive algorithm. This procedure is described in Chapter 4. For such algorithms it is necessary to evaluate the solution in order to obtain estimations of the local error. Appropriate error indicators are presented in Chapter 5.

Generally, refinement of the mesh elements representing the geometrical discretization is done in an isotropic way, by which a two–dimensional quadrilateral is split in half in both dimensions, resulting in four new quadrilaterals. However, flow phenomena exhibit strong directional behavior. In boundary layers or interior layers like shocks, the flow variables change rapidly in the direction orthogonal to the layer, whereas the change in direction of the layer is much smaller. In such cases almost the same accuracy can be achieved by splitting the elements in only one direction, thereby reducing the number of

[1] Of course, this includes costs as well, as the use of hardware for a certain time is generally associated with costs.

elements and accordingly the number of unknowns of the discrete problem. It is the main task of this work to implement an anisotropic refinement capability and to derive and implement appropriate indicators for such an anisotropic refinement. The anisotropic indicators are described in Chapter 6 along with the resulting adaptive mesh refinement strategy. Numerical results for several testcases and a comparison of the performance of different indicators are presented in Chapter 7.
The appendix Chapter A includes some implementational issues associated with anisotropic refinement. This chapter can also be seen as a short reference to the changes implemented in the DG flow solver `PADGE` (Parallel Adaptive Discontinuous Galerkin Environment) currently developed at the DLR Braunschweig and in the `C++` Finite Element library `deal.II` [2] the flow solver is based on.

2. Governing Equations of Computational Fluid Dynamics

Fluid flows can be described by a system of partial differential conservation equations, namely the conservation of mass, momentum and energy. In the case of inviscid flows, the conservation of momentum is given by the compressible Euler equations, whereas in the more general case of a viscous flow it is described by the compressible Navier–Stokes equations. The equation describing the conservation of energy includes viscous terms as well, whereas the continuity equation decribing the conservation of mass is the same for inviscid and viscous flows.
Often the whole set of conservation equations is referred to as the compressible Euler equations and compressible Navier–Stokes equations, respectively. This naming convention is adopted in the following.
In case of the Euler equations, the system of partial differential equations is of hyperbolic type, therefore this type of system is briefly characterized in the following section. Following these general remarks the actual systems considered here, namely the Euler and Navier–Stokes equations, are given in Sections 2.2 and 2.3, respectively.

2.1. Hyperbolic conservation equations

Given a final time $T > 0$, the following system of conservation equations,

$$\begin{aligned} \frac{\partial}{\partial t}\mathbf{u} + \sum_{i=1}^{d} \frac{\partial}{\partial x_i}\mathbf{f}_i^c(\mathbf{u}) = 0 \qquad & \text{in } (0,T] \times \Omega, \\ \mathbf{u}(0,\cdot) = \mathbf{u}_0(\cdot) \quad & \text{in } \Omega, \end{aligned} \tag{2.1}$$

is considered, where Ω is a bounded connected domain in $\mathbb{R}^d$, $d \geq 1$, $\mathbf{u} = (u_1, \ldots, u_m)^T$, $\mathcal{F}^c(\mathbf{u}) = (\mathbf{f}_1^c(\mathbf{u}), \ldots, \mathbf{f}_d^c(\mathbf{u}))$ and $\mathbf{f}_i^c : \mathbb{R}^m \to \mathbb{R}^m$, $i = 1, \ldots, d$, are continuously differentiable. In particular, this work will be concerned with the solution of the *stationary* system of conservation equations,

$$\nabla \cdot \mathcal{F}^c(\mathbf{u}) = 0 \quad \text{in } \Omega, \tag{2.2}$$

subject to appropriate boundary conditions described below. (2.1) is called hyperbolic, if the matrix

$$B(\mathbf{u}, \boldsymbol{\nu}) := \sum_{i=1}^{d} \nu_i A_i(\mathbf{u}) \tag{2.3}$$

has m real eigenvalues and a complete set of linearly independent eigenvectors for all vectors $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_d) \in \mathbb{R}^d$. Here, $A_i(\mathbf{u}) \in \mathbb{R}^{m \times m}$ denotes the Jacobi matrix of the flux $\mathbf{f}_i^c(u)$, i.e.

$$A_i(\mathbf{u}) := \frac{\partial}{\partial \mathbf{u}}\mathbf{f}_i^c(\mathbf{u}), \qquad i = 1, \ldots, d. \tag{2.4}$$

In order to describe a unique problem, the system of conservation equations (2.1) must be supplemented with appropriate boundary conditions; for example at inflow/outflow boundaries, it is required that

$$B^{-}(\mathbf{u},\mathbf{n})\,(\mathbf{u}-\mathbf{g}) = \mathbf{0}, \quad \text{on } \Gamma \tag{2.5}$$

where $\mathbf{n}$ denotes the unit outward normal vector to the boundary $\Gamma = \partial\Omega$ and $\mathbf{g}$ is a (given) vector function. Here, $B^{\pm}(\mathbf{u},\mathbf{n})$ denotes the positive/negative part of $B(\mathbf{u},\mathbf{n})$,

$$B^{\pm}(\mathbf{u},\mathbf{n}) = P\Lambda^{\pm}P^{-1}, \tag{2.6}$$

where $P = [\mathbf{r}_1,\ldots,\mathbf{r}_m]$ denotes the $m\times m$ matrix of eigenvectors $\mathbf{r}_i$ of $B(\mathbf{u},\mathbf{n})$ and $\Lambda^{+} = \mathrm{diag}(\max(\lambda_i,0))$ and $\Lambda^{-} = \mathrm{diag}(\min(\lambda_i,0))$ the $m\times m$ diagonal matrix of the positive/negative eigenvalues λ_i of $B(\mathbf{u},\mathbf{n})$ with $B\mathbf{r}_i = \lambda_i\mathbf{r}_i$, $i = 1,\ldots,d$.

2.2. Euler equations

The Euler equations of compressible gas dynamics describing an inviscid fluid flow represent an important example of the hyperbolic problem (2.1). In two space–dimensions, the vector of conservative variables $\mathbf{u}$ and the convective fluxes $\mathbf{f}_i^c$, $i = 1,2$, are defined by

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ \rho E \end{bmatrix}, \quad \mathbf{f}_1^c(\mathbf{u}) = \begin{bmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ \rho H v_1 \end{bmatrix} \quad \text{and} \quad \mathbf{f}_2^c(\mathbf{u}) = \begin{bmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ \rho H v_2 \end{bmatrix}, \tag{2.7}$$

where ρ, $\mathbf{v} = (v_1,v_2)^T$, p and E denote the density, velocity vector, pressure and specific total energy, respectively. Additionally, H is the total enthalpy given by

$$H = E + \frac{p}{\rho} = e + \tfrac{1}{2}\mathbf{v}^2 + \frac{p}{\rho}, \tag{2.8}$$

where e is the specific static internal energy, and the pressure is determined by the equation of state of an ideal gas

$$p = (\gamma-1)\rho e, \tag{2.9}$$

where $\gamma = c_p/c_v$ is the ratio of specific heat capacities at constant pressure, c_p, and constant volume, c_v; for dry air at moderate temperatures, $\gamma = 1.4$, the theoretical value for duatomic gases from the kinetic theory of gases, is a good approximation to values obtained by measurements. The flux Jacobians $A_i(\mathbf{u})$ defined in (2.4) are given by

$$A_1(\mathbf{u}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -v_1^2 + \frac{1}{2}(\gamma-1)\mathbf{v}^2 & (3-\gamma)v_1 & -(\gamma-1)v_2 & \gamma-1 \\ -v_1v_2 & v_2 & v_1 & 0 \\ v_1\left(\frac{1}{2}(\gamma-1)\mathbf{v}^2 - H\right) & H-(\gamma-1)v_1^2 & -(\gamma-1)v_1v_2 & \gamma v_1 \end{pmatrix},$$

$$A_2(\mathbf{u}) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -v_1v_2 & v_2 & v_1 & 0 \\ -v_2^2 + \frac{1}{2}(\gamma-1)\mathbf{v}^2 & -(\gamma-1)v_1 & (3-\gamma)v_2 & \gamma-1 \\ v_2\left(\frac{1}{2}(\gamma-1)\mathbf{v}^2 - H\right) & -(\gamma-1)v_1v_2 & H-(\gamma-1)v_2^2 & \gamma v_2 \end{pmatrix}.$$

Finally, the eigenvalues of the matrix $B(\mathbf{u},\mathbf{n}) = \sum_{i=1}^{2} n_i A_i(\mathbf{u})$ are

$$\lambda_1 = \mathbf{v}\cdot\mathbf{n} - c, \quad \lambda_2 = \lambda_3 = \mathbf{v}\cdot\mathbf{n}, \quad \lambda_4 = \mathbf{v}\cdot\mathbf{n} + c \tag{2.10}$$

where $c = \sqrt{\gamma p/\rho}$ denotes the speed of sound. Considering the signs of λ_i, $i = 1,\ldots,4$, we distinguish four cases of boundary conditions (2.5):

- supersonic inflow: $\lambda_i < 0$, $i = 1,\ldots,4$,
- subsonic inflow: $\lambda_i < 0$, $i = 1,2,3$, $\lambda_4 > 0$,
- subsonic outflow: $\lambda_1 < 0$, $\lambda_i > 0$, $i = 2,3,4$, and
- supersonic outflow: $\lambda_i > 0$, $i = 1,\ldots,4$.

A fifth case of boundary conditions can be found at walls, where (at least) the normal component of the flow vanishes: $\mathbf{v}\cdot\mathbf{n} = 0$, resulting in the following eigenvalues:

- wall: $\lambda_1 = -c < 0, \lambda_2 = \lambda_3 = 0, \lambda_4 = c > 0$.

Each eigenvalue smaller than zero corresponds to an inflow characteristic. The number of variables to be prescribed on the boundary depends on the number of inflow characteristics.

2.3. Navier–Stokes equations

The compressible Euler equations can be extended to viscous flows by considering additional viscous fluxes. This leads to the compressible Navier–Stokes equations, which will be given for the two–dimensional steady state case below.
Like in Section 2.2, ρ, $\mathbf{v} = (v_1, v_2)^T$, p and E denote the density, velocity vector, pressure and specific total energy, respectively. Furthermore, T denotes the temperature. The equations of motion are given by

$$\nabla\cdot(\mathcal{F}^c(\mathbf{u}) - \mathcal{F}^v(\mathbf{u},\nabla\mathbf{u})) \equiv \frac{\partial}{\partial x_i}\mathbf{f}_i^c(\mathbf{u}) - \frac{\partial}{\partial x_i}\mathbf{f}_i^v(\mathbf{u},\nabla\mathbf{u}) = 0 \quad \text{in } \Omega. \tag{2.11}$$

The vector of conservative variables $\mathbf{u}$ and the convective fluxes $\mathbf{f}_i^c$, $i = 1,2$, are given by (2.7). Furthermore, the viscous fluxes $\mathbf{f}_i^v$, $i = 1,2$, are defined by

$$\mathbf{f}_1^v(\mathbf{u},\nabla\mathbf{u}) = \begin{bmatrix} 0 \\ \tau_{11} \\ \tau_{21} \\ \tau_{1j}v_j + \mathcal{K}T_{x_1} \end{bmatrix} \quad \text{and} \quad \mathbf{f}_2^v(\mathbf{u},\nabla\mathbf{u}) = \begin{bmatrix} 0 \\ \tau_{12} \\ \tau_{22} \\ \tau_{2j}v_j + \mathcal{K}T_{x_2} \end{bmatrix}, \tag{2.12}$$

respectively, where $\mathcal{K}$ is the thermal conductivity coefficient. It is obvious, that the conservation of mass, which is described by the first component of the system, is not influenced by viscous terms.
Finally, for a Newtonian fluid the viscous stress tensor is defined by

$$\tau = \mu\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T - \tfrac{2}{3}(\nabla\cdot\mathbf{v})I\right), \tag{2.13}$$

where μ is the dynamic viscosity coefficient, and the temperature T is given by $e = c_v T$; thus

$$\mathcal{K}T = \tfrac{\mu\gamma}{Pr}\left(E - \tfrac{1}{2}\mathbf{v}^2\right),$$

where $Pr = \frac{\mu c_p}{\mathcal{K}}$ is the Prandtl number and $Pr = 0.72$ should be chosen for dry air at moderate temperatures.
For the purpose of discretization, the compressible Navier–Stokes equations (2.11) can be rewritten in the following (equivalent) form:

$$\frac{\partial}{\partial x_i}\left(\mathbf{f}_i^c(\mathbf{u}) - G_{ij}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x_j}\right) = 0 \quad \text{in } \Omega. \tag{2.14}$$

Here, the matrices $G_{ij}(\mathbf{u}) = \partial \mathbf{f}_i^v(\mathbf{u}, \nabla\mathbf{u})/\partial u_{x_j}$, for $i, j = 1, 2$ are the Jacobian matrices of the viscous fluxes with respect to the conservation variables, i.e., $\mathbf{f}_i^v(\mathbf{u}, \nabla\mathbf{u}) = G_{ij}(\mathbf{u})\partial\mathbf{u}/\partial x_j$, $i = 1, 2$, where

$$G_{11} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -\frac{4}{3}v_1 & \frac{4}{3} & 0 & 0 \\ -v_2 & 0 & 1 & 0 \\ -\left(\frac{4}{3}v_1^2 + v_2^2 + \frac{\gamma}{Pr}\left(E - \mathbf{v}^2\right)\right) & \left(\frac{4}{3} - \frac{\gamma}{Pr}\right)v_1 & \left(1 - \frac{\gamma}{Pr}\right)v_2 & \frac{\gamma}{Pr} \end{pmatrix},$$

$$G_{12} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{2}{3}v_2 & 0 & -\frac{2}{3} & 0 \\ -v_1 & 1 & 0 & 0 \\ -\frac{1}{3}v_1v_2 & v_2 & -\frac{2}{3}v_1 & 0 \end{pmatrix}, G_{21} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -v_2 & 0 & 1 & 0 \\ \frac{2}{3}v_1 & -\frac{2}{3} & 0 & 0 \\ -\frac{1}{3}v_1v_2 & -\frac{2}{3}v_2 & v_1 & 0 \end{pmatrix},$$

$$G_{22} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -v_1 & 1 & 0 & 0 \\ -\frac{4}{3}v_2 & 0 & \frac{4}{3} & 0 \\ -\left(v_1^2 + \frac{4}{3}v_2^2 + \frac{\gamma}{Pr}\left(E - \mathbf{v}^2\right)\right) & \left(1 - \frac{\gamma}{Pr}\right)v_1 & \left(\frac{4}{3} - \frac{\gamma}{Pr}\right)v_2 & \frac{\gamma}{Pr} \end{pmatrix}.$$

Given that $\Omega \subset \mathbb{R}^2$ is a bounded region, with boundary Γ, the system of conservation equations (2.14) must be supplemented by appropriate boundary conditions. It must be distinguished between supersonic inflow (Dirichlet), subsonic inflow, subsonic outflow, supersonic outflow (Neumann) and solid wall boundaries, denoted by $\Gamma_{\text{D,sup}}$, $\Gamma_{\text{D,sub-in}}$, $\Gamma_{\text{D,sub-out}}$, Γ_{N} and Γ_{W}, respectively. Additional information on these boundary conditions can be found in Section 3.5.
For solid wall boundaries and *viscous* flows, *isothermal* and *adiabatic* conditions can be distinguished. To this end, decomposing $\Gamma_W = \Gamma_{\text{W,iso}} \cup \Gamma_{\text{W,adia}}$, the boundary conditions read

$$\mathbf{v} = \mathbf{0} \;\text{ on }\; \Gamma_{\text{W}}, \qquad T = T_{\text{wall}} \;\text{ on }\; \Gamma_{\text{W,iso}}, \qquad \mathbf{n}\cdot\nabla T = 0 \;\text{ on }\; \Gamma_{\text{W,adia}}, \tag{2.15}$$

where T_{wall} is a given wall temperature.
For solid wall boundaries and *inviscid* flows, governed by the compressible Euler equations, see Section 2.2 or equation (2.11) with $\mu = 0$, *reflective* (or slip wall) boundary conditions given by

$$\mathbf{v}\cdot\mathbf{n} = 0 \;\text{ on }\; \Gamma_{\text{W}} = \Gamma_{\text{refl}} \tag{2.16}$$

are considered.

3. Discontinuous Galerkin methods

3.1. Discontinuous function spaces

Numerical methods are based on discretizations of continuous problems. Only after these discretizations have been applied to the problem, the approximate solution can be obtained. Concerning the process of discretization, two general concepts have to be distinguished: *geometrical discretization* and *discretization of the function space*.
Geometrical discretization has to be applied to the domain Ω on which the problem is defined. For Finite Difference methods this discretization is based on distributing nodes in the domain, for Finite Volume and Finite Element methods the domain is decomposed into elements[2]. As Discontinous Galerkin Methods are a special type of Finite Element methods (FE methods for short), only those will be considered further. In general, the elements have simple geometric shape, for example triangles or quadrilaterals in two dimensions and tetrahedra, pyramids, prisms or hexahedra in three dimensions. These elements cover the whole domain and should be non–overlapping.
Based on this geometrical discretization, the solution is approximated piecewise on each element. In order to express the continuous solution with a finite number of unknowns (degrees of freedom), the approximation is restricted to a certain function space, usually the space of polynomials up to a certain maximum degree. Varying the geometrical discretization and the discrete function space, the solution can be approximated with varying accuracy.
"Standard" (or continuous) FE methods demand, that the discrete solution is continuous over the edges and faces of all elements. This convention effectively reduces the total number of degrees of freedom, as the degrees of freedom of neighboring elements are directly coupled. Clearly, the nature of this coupling depends on the polynomials chosen as a basis.
The basic idea of Discontinuous Galerkin methods is to avoid the restriction of continuity. Allowing the discrete solution to jump over the edges and faces of elements, the local approximation can be improved. Furthermore it is easier to construct locally refined geometrical discretizations (meshes), as there is theoretically no restriction on the number of *hanging nodes*, where the face of one element can be neighbored by several other elements. Whereas hanging nodes require no special treatment in DG methods, they do so in continuous FE methods. There, the discrete solution on the finer side has to be restricted to the coarser discretization along the edge or face. This results in the need of restrictive modifications in the assembled global matrices.
The degrees of freedom have to be coupled between neighboring elements for DG methods as well. However, this is not a direct coupling like in continuous FE methods. Instead, DG methods use an indirect coupling realized by inter–element face terms including *numerical flux functions* and *interior penalty terms*. These concepts will be explained in the following sections. Before these details are given, some general remarks about

[2]Here and in the following the term *element* refers to the entities of the geometrical subdivision of a domain as well as to these entities associated with a piecewise function space.

the advantages of Discontinuous Galerkin methods will be presented. This summary of important aspects is inspired by Houston *et al.* [17]:

- DG methods for flow calculations are, by construction, locally conservative. As the governing equations are conservation equations, namely the conservation of mass, energy and momentum, the numerical method ensures an important characteristic of the problem in a natural way.
- DG methods exhibit enhanced stability properties in the vicinity of interior or boundary layers. Oscillations near discontinuities, for example, are limited to the elements in the direct vicinity of these phenomena.
- The indirect coupling between degrees of freedom from neighboring elements allows to use irregular grids. Restrictions on the mesh regularity are based on implementational aspects only, not on numerical reasons.
- Furthermore, in the context of hp–refinement (see Section 4.1.3) the polynomial degree on a single element can be chosen without restrictions induced by the neighboring elements.

However, there are also some disadvantages. The most important one is the increased number of degrees of freedom: on a structured mesh using quadrilateral elements with a continuous piecewise bilinear approximation, the number of elements is almost identical to the number of degrees of freedom, the latter being only slightly higher due to boundary effects. For a discontinuous bilinear approximation on the same mesh the number of degrees of freedom is four times the number of elements and thereby almost four times as high as in the continuous case. This ratio reduces for higher polynomial degrees, but nevertheless it represents the major drawback of applying DG methods. There are some first attempts in literature of combining DG and continuous FE methods, combining the advantage of enhanced stability of DG methods in those regions of the flow, which might need this property, with the reduced number of degrees of freedom in the case of continuous FE methods in other parts of the domain. However, the following part of this work will only be concerned with "pure" DG methods.

3.2. Discretization and notation

Before the discretization of flow equations can be given, some notation has to be introduced. In the following it is assumed, that Ω can be subdivided into shape-regular meshes $\mathcal{T}_h = \{\kappa\}$ consisting of quadrilateral elements κ. Here, h denotes the piecewise constant mesh function defined by $h|_\kappa \equiv h_\kappa = \mathrm{diam}(\kappa)$ for all $\kappa \in \mathcal{T}_h$. It can further be assumed, that each $\kappa \in \mathcal{T}_h$ is an image of a fixed reference element $\hat{\kappa}$, that is, $\kappa = \sigma_\kappa(\hat{\kappa})$ for all $\kappa \in \mathcal{T}_h$. In the following, only the case when $\hat{\kappa}$ is the open unit square in $\mathbb{R}^2$ and the open unit cube in $\mathbb{R}^3$, respectively, will be considered. Furthermore the mapping σ_κ of the reference element $\hat{\kappa}$ to the element κ in real space is assumed to be bijective and smooth, with the eigenvalues of its Jacobian matrix being bounded from below and above. For elements in the interior of the domain, $\partial\kappa \cap \Gamma = \emptyset$, the mapping σ_κ is given by a d-linear function. In order to represent curved boundaries, see Figure 3.1, mappings can be used that include polynomials of higher degree on boundary elements. In both cases the mapping σ_κ can be decomposed into a linear part $\bar{\sigma}_\kappa$ and a higher order part $\tilde{\sigma}_\kappa$, such that $\kappa = \sigma_\kappa(\hat{\kappa}) = \tilde{\sigma}_\kappa\left(\bar{\sigma}_\kappa(\hat{\kappa})\right)$.

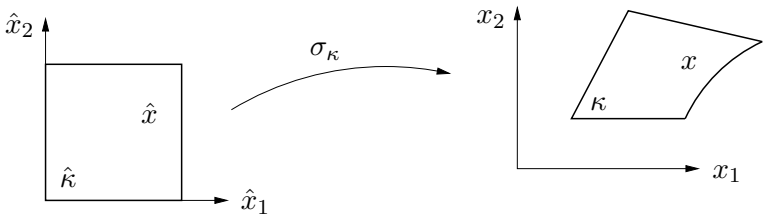


Figure 3.1.: Mapping σ_κ of reference element $\hat{\kappa}$ to the element κ in real space.

On the reference element $\hat{\kappa}$ spaces of tensor product polynomials of degree $p \geq 0$ are defined as follows:

$$\mathcal{Q}_p(\hat{\kappa}) = \operatorname{span}\left\{\hat{x}^\alpha : 0 \leq \alpha_i \leq p,\ 0 \leq i \leq d\right\}, \tag{3.1}$$

where α denotes a multi-index and $\hat{x}^\alpha = \prod_{i=1}^d \hat{x}_i^{\alpha_i}$. Finally, the finite element space $\mathbf{V}_h^p$ consisting of discontinuous vector–valued piecewise polynomial functions of degree $p \geq 0$ is defined by

$$\mathbf{V}_h^p = \{\mathbf{v}_h \in [L_2(\Omega)]^m : \mathbf{v}_h|_\kappa \circ \sigma_\kappa \in [\mathcal{Q}_p(\hat{\kappa})]^m\}. \tag{3.2}$$

Suppose that $\mathbf{v}|_\kappa \in \left[H^1(\kappa)\right]^m$ for each $\kappa \in \mathcal{T}_h$. Let κ and κ' be two adjacent elements of $\mathcal{T}_h$ and $\mathbf{x}$ be an arbitrary point on the interior edge $e = \partial\kappa \cap \partial\kappa'$. $\mathbf{v}_\kappa^\pm$ denote the traces of $\mathbf{v}$ taken from within the interior of κ and κ', respectively, see Figure 3.2. For $\mathbf{x} \in \partial\kappa \cap \Gamma$ the outer trace is set to be $\mathbf{v}_\kappa^- := \mathbf{g}$ where $\mathbf{g}$ denotes an appropriate boundary function.

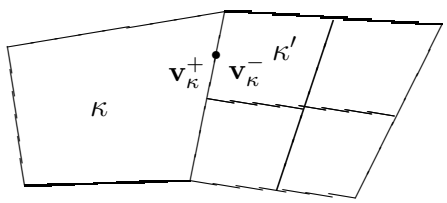


Figure 3.2.: Definition of the interior and outer traces $\mathbf{v}_\kappa^\pm$ wrt. element κ.

In the following it will always be clear from the context which element κ in the subdivision $\mathcal{T}_h$ the quantities $\mathbf{v}_\kappa^+$ and $\mathbf{v}_\kappa^-$ correspond to, therefore the letter κ in the subscript will be suppressed for the sake of notational simplicity, leading to the notation $\mathbf{v}^+$ and $\mathbf{v}^-$.

3.3. Discretization of the compressible Euler equations

To formulate the discontinuous Galerkin method, as first step the weak formulation of (2.2) is introduced. To this end, Equation (2.2) is multiplied by an arbitrary smooth (vector-)function $\mathbf{v}$ and integrated by parts over each element κ in the mesh $\mathcal{T}_h$. After summation over all elements $\kappa \in \mathcal{T}_h$ this leads to

$$\sum_{\kappa \in \mathcal{T}_h} \left\{ -\int_\kappa \mathcal{F}^c(\mathbf{u}) : \nabla \mathbf{v} \, \mathrm{d}\mathbf{x} + \int_{\partial\kappa} \mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n}\, \mathbf{v} \, \mathrm{d}s \right\} = 0. \tag{3.3}$$

To discretize (3.3), the analytical solution $\mathbf{u}$ is replaced by the Galerkin Finite Element approximation $\mathbf{u}_h$ and the test function $\mathbf{v}$ by $\mathbf{v}_h$, where $\mathbf{u}_h$ and $\mathbf{v}_h$ both belong to the finite element space $\mathbf{V}_h^p$. In addition, since the numerical solution $\mathbf{u}_h$ is discontinuous between element interfaces, the flux $\mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n}$ must be replaced by a *numerical flux* function $\mathcal{H}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n})$, which depends on both the interior– and outer–trace of $\mathbf{u}_h$ on $\partial\kappa$, $\kappa \in \mathcal{T}_h$, and the unit outward normal $\mathbf{n}$ to $\partial\kappa$. Thereby the Discontinuous Galerkin Finite Element discretization of (2.2) is given as follows: find $\mathbf{u}_h \in \mathbf{V}_h^p$ such that

$$\sum_{\kappa \in \mathcal{T}_h} \left\{ -\int_\kappa \mathcal{F}^c(\mathbf{u}_h) : \nabla \mathbf{v}_h \, \mathrm{d}\mathbf{x} + \int_{\partial\kappa} \mathcal{H}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n}) \, \mathbf{v}_h^+ \, \mathrm{d}s \right\} = 0 \quad \forall \mathbf{v}_h \in \mathbf{V}_h^p. \tag{3.4}$$

This scheme is called discontinuous Galerkin method of degree p, or in short, "DG(p) method".
For elements $\kappa \in \mathcal{T}_h$ whose boundaries intersect that of the computational domain Ω, $\mathbf{u}_h^-$ has to be replaced by appropriate boundary conditions on the portion of $\partial\kappa$ for which $\partial\kappa \cap \Gamma \neq \emptyset$. For more details about boundary conditions for the compressible Euler and Navier-Stokes equations, see Section 3.4.
The replacement of the flux $\mathcal{F}^c(\mathbf{u}) \cdot \mathbf{n}$ by the numerical flux function $\mathcal{H}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n})$ on the boundary of element κ, κ in $\mathcal{T}_h$, corresponds to the weak imposition of the boundary data.
It should be noted, that the choice of a numerical flux function is independent of the finite element space employed. Indeed, the numerical flux $\mathcal{H}(\cdot,\cdot,\cdot)$ may be chosen to be any two–point monotone Lipschitz function which satisfies the following two conditions:

(i) $\mathcal{H}(\cdot,\cdot,\cdot)|_{\partial\kappa}$ is consistent with the flux $\mathcal{F}^c(\cdot) \cdot \mathbf{n}$ for each κ in $\mathcal{T}_h$; i.e.

$$\mathcal{H}(\mathbf{v}, \mathbf{v}, \mathbf{n})|_{\partial\kappa} = \mathcal{F}^c(\mathbf{v}) \cdot \mathbf{n} \quad \forall \kappa \in \mathcal{T}_h;$$

(ii) $\mathcal{H}(\cdot,\cdot,\cdot)$ is conservative, i.e. given any two neighbouring elements κ and κ' from the finite element partition $\mathcal{T}_h$, at each point $x \in \partial\kappa \cap \partial\kappa' \neq \emptyset$, noting that $\mathbf{n}_{\kappa'} = -\mathbf{n}$, it holds that

$$\mathcal{H}(\mathbf{v}, \mathbf{w}, \mathbf{n}) = -\mathcal{H}(\mathbf{w}, \mathbf{v}, -\mathbf{n}).$$

There are several numerical flux functions satisfying these conditions, such as the Godunov, Engquist–Osher, Lax–Friedrichs, Roe or the Vijayasundaram flux, for example. As examples, two different numerical fluxes are considered here: the (local) Lax–Friedrichs flux and the Vijayasundaram flux.
The **(local) Lax–Friedrichs flux** $\mathcal{H}_{LF}(\cdot,\cdot,\cdot)$, is defined by

$$\mathcal{H}_{LF}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n})|_{\partial\kappa} = \tfrac{1}{2} \left(\mathcal{F}^c(\mathbf{u}^+) \cdot \mathbf{n} + \mathcal{F}^c(\mathbf{u}^-) \cdot \mathbf{n} + \alpha \left(\mathbf{u}^+ - \mathbf{u}^- \right) \right), \tag{3.5}$$

for $\kappa \in \mathcal{T}_h$, where α is the maximum over $\mathbf{u}^+$ and $\mathbf{u}^-$,

$$\alpha = \alpha\left(\mathbf{u}^+, \mathbf{u}^-\right) = \max_{\mathbf{v}=\mathbf{u}^+,\mathbf{u}^-} \{|\lambda(B(\mathbf{v}, \mathbf{n}))|\}, \tag{3.6}$$

of the largest eigenvalue (in absolute value) $|\lambda(B)|$ of the matrix $B(\mathbf{v}, \mathbf{n}) = \sum_{i=0}^d n_i A_i(\mathbf{u})$ defined in (2.3).

The **Vijayasundaram flux** $\mathcal{H}_V(\cdot,\cdot,\cdot)$, is defined by

$$\mathcal{H}_V(\mathbf{u}^+,\mathbf{u}^-,\mathbf{n})|_{\partial\kappa} = B^+(\bar{\mathbf{u}},\mathbf{n})\mathbf{u}^+ + B^-(\bar{\mathbf{u}},\mathbf{n})\mathbf{u}^- \quad \text{for } \kappa \in \mathcal{T}_h, \tag{3.7}$$

where $B^+(\bar{\mathbf{u}},\mathbf{n})$ and $B^-(\bar{\mathbf{u}},\mathbf{n})$ denote the positive and negative parts, cf. (2.6), of the matrix $B(\bar{\mathbf{u}},\mathbf{n})$, respectively, evaluated at an average state $\bar{\mathbf{u}}$ between $\mathbf{u}^+$ and $\mathbf{u}^-$.

3.4. Discretization of the compressible Navier–Stokes equations

In the following, the interior penalty (IP) discontinuous Galerkin discretization of the compressible Navier-Stokes equations according to Hartmann and Houston [14, 15] is introduced. In addition to the notation introduced in the previous sections, average and jump operators for vector- and matrix-valued functions have to be defined. To this end, let κ^+ and κ^- be two adjacent elements of $\mathcal{T}_h$ and $\mathbf{x}$ be an arbitrary point on the interior edge $e = \partial\kappa^+ \cap \partial\kappa^- \subset \Gamma_{\mathcal{I}}$. Moreover, let $\mathbf{v}$ and $\underline{\tau}$ be vector- and matrix-valued functions, respectively, that are smooth inside each element $\kappa \in \mathcal{T}_h$. Superscripts $\pm$ in $\mathbf{v}^\pm := \mathbf{v}|_{\partial\kappa^\pm}$ and $\underline{\tau}^\pm := \underline{\tau}|_{\partial\kappa^\pm}$ denote the traces of, respectively, $\mathbf{v}$ and $\underline{\tau}$ on e taken from within the interior of $\kappa^\pm$. Then, the averages at $\mathbf{x} \in e$ are defined by

$$\{\mathbf{v}\} = (\mathbf{v}^+ + \mathbf{v}^-)/2 \quad \text{and} \quad \{\underline{\tau}\} = (\underline{\tau}^+ + \underline{\tau}^-)/2. \tag{3.8}$$

Similarly, the jump at $\mathbf{x} \in e$ is given by

$$\underline{[\![\mathbf{v}]\!]} = \mathbf{v}^+ \otimes \mathbf{n}_{\kappa^+} + \mathbf{v}^- \otimes \mathbf{n}_{\kappa^-}. \tag{3.9}$$

On a boundary edge $e \subset \Gamma$, the operators are set to $\{\mathbf{v}\} = \mathbf{v}$, $\{\underline{\tau}\} = \underline{\tau}$ and $\underline{[\![\mathbf{v}]\!]} = \mathbf{v} \otimes \mathbf{n}$. Note, that for the jump and average operators the boundary function $\mathbf{g}$ is not involved. For matrices $\underline{\sigma}, \underline{\tau} \in \mathbb{R}^{m\times n}$, $m, n \geq 1$, the standard notation $\underline{\sigma} : \underline{\tau} = \sum_{k=1}^m \sum_{l=1}^n \sigma_{kl}\tau_{kl}$ is used; additionally, for vectors $\mathbf{v} \in \mathbb{R}^m, \mathbf{w} \in \mathbb{R}^n$, the matrix $\mathbf{v} \otimes \mathbf{w} \in \mathbb{R}^{m\times n}$ is defined by $(\mathbf{v} \otimes \mathbf{w})_{kl} = v_k\, w_l$.
According to [14] the interior penalty discontinuous Galerkin discretization of the compressible Navier–Stokes equations (2.11) is given by: find $\mathbf{u}_h \in \mathbf{V}_h^p$ such that

$$\begin{aligned}
\mathcal{N}(\mathbf{u}_h,\mathbf{v}) \;\equiv\; & -\int_\Omega \mathcal{F}^c(\mathbf{u}_h) : \nabla_h \mathbf{v}\,\mathrm{d}\mathbf{x} + \sum_{\kappa\in\mathcal{T}_h} \int_{\partial\kappa\setminus\Gamma} \mathcal{H}(\mathbf{u}_h^+,\mathbf{u}_h^-,\mathbf{n}^+)\cdot\mathbf{v}^+\,\mathrm{d}s \\
& + \int_\Omega \mathcal{F}^v(\mathbf{u}_h,\nabla_h\mathbf{u}_h) : \nabla_h\mathbf{v}\,\mathrm{d}\mathbf{x} - \int_{\Gamma_{\mathcal{I}}} \{\mathcal{F}^v(\mathbf{u}_h,\nabla_h\mathbf{u}_h)\} : \underline{[\![\mathbf{v}]\!]}\,\mathrm{d}s \\
& + \int_{\Gamma_{\mathcal{I}}} \theta\{G^\top(\mathbf{u}_h)\nabla_h\mathbf{v}\} : \underline{[\![\mathbf{u}_h]\!]}\,\mathrm{d}s + \int_{\Gamma_{\mathcal{I}}} \delta\underline{[\![\mathbf{u}_h]\!]} : \underline{[\![\mathbf{v}]\!]}\,\mathrm{d}s + \mathcal{N}_\Gamma(\mathbf{u}_h,\mathbf{v}) = 0
\end{aligned} \tag{3.10}$$

for all $\mathbf{v}$ in $\mathbf{V}_h^p$, where δ denotes the discontinuity penalization matrix. One possible choice is setting $\delta = \mathrm{diag}\{\delta_i, i = 1,\ldots,4\}$, where

$$\delta_i|_e = C_{\mathrm{IP}}\frac{\mu p^2}{\tilde{h}} \quad \text{for } e \subset \Gamma_{\mathcal{I}} \cup \Gamma, \tag{3.11}$$

$\tilde{h} = \min(\mathrm{meas}(\kappa), \mathrm{meas}(\kappa'))/\mathrm{meas}(e)$ represents the element dimension orthogonal to

the edge e of elements κ and κ' adjacent to e, and C_{IP} is a positive constant, which, for reasons of stability, must be chosen sufficiently large[3].
Finally, the boundary terms included in $\mathcal{N}_\Gamma(\mathbf{u}_h, \mathbf{v})$ are given by

$$\begin{aligned}\mathcal{N}_\Gamma(\mathbf{u}_h, \mathbf{v}) = &\int_\Gamma \mathcal{H}_\Gamma(\mathbf{u}_h^+, \mathbf{u}_\Gamma(\mathbf{u}_h^+), \mathbf{n}^+) \cdot \mathbf{v}^+ \,\mathrm{d}s + \int_\Gamma \delta\left(\mathbf{u}_h^+ - \mathbf{u}_\Gamma(\mathbf{u}_h^+)\right) \cdot \mathbf{v}^+ \,\mathrm{d}s, \\ &- \int_\Gamma \mathbf{n} \cdot \mathcal{F}_\Gamma^v(\mathbf{u}_h^+, \nabla_h \mathbf{u}_h^+)\, \mathbf{v}^+ \,\mathrm{d}s \\ &+ \theta \int_\Gamma \left(G_\Gamma^\top(\mathbf{u}_h^+) \nabla_h \mathbf{v}_h^+\right) : \left(\mathbf{u}_h^+ - \mathbf{u}_\Gamma(\mathbf{u}_h^+)\right) \otimes \mathbf{n} \,\mathrm{d}s.\end{aligned} \tag{3.12}$$

Here, the viscous boundary flux $\mathcal{F}_\Gamma^v$ and the corresponding homogeneity tensor G_Γ are defined by

$$\mathcal{F}_\Gamma^v(\mathbf{u}_h, \nabla \mathbf{u}_h) = \mathcal{F}^v(\mathbf{u}_\Gamma(\mathbf{u}_h), \nabla \mathbf{u}_h) = G_\Gamma(\mathbf{u}_h) \nabla \mathbf{u}_h = G(\mathbf{u}_\Gamma(\mathbf{u}_h)) \nabla \mathbf{u}_h. \tag{3.13}$$

Furthermore, on adiabatic boundaries $\Gamma_{\text{adia}} \subset \Gamma_W$, $\mathcal{F}_\Gamma^v$ and G_Γ are modified such that $\mathbf{n} \cdot \nabla T = 0$. For an adjoint consistent discretization of boundary terms, see [11, 12], a numerical boundary flux function $\mathcal{H}_\Gamma$ is defined by

$$\mathcal{H}_\Gamma(\mathbf{u}_h^+, \mathbf{u}_\Gamma(\mathbf{u}_h^+), \mathbf{n}) = \mathbf{n} \cdot \mathcal{F}_\Gamma^c(\mathbf{u}_h^+) = \mathbf{n} \cdot \mathcal{F}^c(\mathbf{u}_\Gamma(\mathbf{u}_h^+)), \tag{3.14}$$

where the boundary function $\mathbf{u}_\Gamma(\cdot)$ will be defined in Section 3.5.

3.5. Boundary conditions

The boundary function $\mathbf{u}_\Gamma(\mathbf{u})$ is given according to the type of boundary condition imposed. In detail:

$$\mathbf{u}_\Gamma(\mathbf{u}) = \mathbf{g}_D \quad \text{on} \quad \Gamma_{\text{D,sup}}, \qquad \mathbf{u}_\Gamma(\mathbf{u}) = \mathbf{u} \quad \text{on} \quad \Gamma_{\text{N}},$$

$$\mathbf{u}_\Gamma(\mathbf{u}) = \left((g_D)_1, (g_D)_2, (g_D)_3, \frac{p(\mathbf{u})}{\gamma - 1} + \frac{(g_D)_2^2 + (g_D)_3^2}{2(g_D)_1}\right)^T \quad \text{on} \quad \Gamma_{\text{D,sub-in}}, \tag{3.15}$$

and

$$\mathbf{u}_\Gamma(\mathbf{u}) = \left(u_1, u_2, u_3, \frac{p_{\text{out}}}{\gamma - 1} + \frac{u_2^2 + u_3^2}{2u_1}\right)^T \quad \text{on} \quad \Gamma_{\text{D,sub-out}}. \tag{3.16}$$

Here, $p \equiv p(\mathbf{u})$ denotes the pressure evaluated using the equation of state (2.9). For viscous flows, set

$$\mathbf{u}_\Gamma(\mathbf{u}) = (u_1, 0, 0, u_1 c_v T_{\text{wall}})^T \quad \text{on} \quad \Gamma_{\text{W,iso}},$$

and

$$\mathbf{u}_\Gamma(\mathbf{u}) = (u_1, 0, 0, u_4)^T \quad \text{on} \quad \Gamma_{\text{W,adia}}.$$

[3] For a wide range of problems $C_{\text{IP}} = 10$ is a good choice.

Finally, for inviscid flows, set

$$\mathbf{u}_\Gamma(\mathbf{u}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-n_1^2 & -n_1 n_2 & 0 \\ 0 & -n_1 n_2 & 1-n_2^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{u} \quad \text{on} \quad \Gamma_W, \tag{3.17}$$

which originates from $\mathbf{u}$ by subtracting the normal velocity component of $\mathbf{u}$, i.e. $\mathbf{v} = (v_1, v_2)$ is replaced by $\mathbf{v}_\Gamma = \mathbf{v} - (\mathbf{v} \cdot \mathbf{n})\mathbf{n}$ which ensures that the normal velocity component vanishes, $\mathbf{v}_\Gamma \cdot \mathbf{n} = 0$.

4. Mesh refinement

4.1. Mesh refinement techniques

Numerical methods yield approximate solutions to complex problems. Compared to the exact solution $\mathbf{u}$ of the problem, the approximated one $\mathbf{u}_h$ contains an error $\mathbf{e}^{(h)} = \mathbf{u} - \mathbf{u}_h$. This error depends on the numerical method used to gain the approximate solution, but also on characteristics of the problem, the triangulation $\mathcal{T}_h$, and in particular its local mesh size h. Uniformly reducing the mesh size of the whole triangulation reduces the error. The asymptotic rate, at which the error is reduced depends on the chosen method. Applications of CFD to real world tasks often ask for a certain accuracy in the obtained results[4]. In order to minimize the error, the mesh has to be sufficiently fine, at least in those regions which would otherwise create large errors. It is not an easy task, however, to design an optimal mesh *a priori*, i.e. without knowledge of the solution. Even if certain characteristics of the solution were known, or else it could be guessed, where the elements should be small and where they can be larger, it is almost never possible to decide, prior to the solution process, how many elements to use to achieve a certain accuracy. Using a fixed mesh will therefore most likely result in either an overusage of resources, if the selected mesh is too fine, or otherwise in an inaccurate solution, if the selected mesh is too coarse. Because of these shortcomings of using a fixed mesh, most applications rely on an iterative procedure, which uses several discretizations with increasing complexity until the desired accuray is achieved. There are several general techniques, which can be used to modify the dicretization and which will be described shortly in the following sections.

4.1.1. h–refinement

Probably the easiest way of obtaining a discretization of a higher complexity is to increase the number of elements and thereby unknowns by splitting each edge in half and forming new elements. Several new child elements replace one mother element, the ratio of children to mother elements depends on the dimension d of the domain and is given by 2^d for the reference elements under consideration. Along with the elements, the number of unknowns will increase in a very similar way, for DG methods the factor is exactly the same. As this procedure reduces the mesh size h, it is called h–refinement.
The function space of the old discretization is completely embedded in the newly created one, thus an interpolation of the approximate solution from the old to the new disretization is an easy task and can be done without loss of information. This ensures optimal initial conditions for the approximate solution, if an iterative scheme is used to solve the discretized equations.

[4] In contrast to that, scientific tasks often ask for the maximum accuracy that can be obtained using given resources. The problems and corresponding approaches described below remain mainly the same, however.

4.1.2. p–refinement

If the numerical method employed allows to use different orders of the piecewise polynomial approximation, which is the case for spectral methods or DG methods, a richer function space for the solution can be obtained using the same triangulation $\mathcal{T}_h$, but a higher polynomial degree. As this polynomial degree is generally abbreviated by p, the procedure is known as p–refinement.
In common with h–refinement it has the positive property of using embedded function spaces, so that the interpolation from the coarser to the refined discretization is lossless except for numerical inaccuracies. As a higher polynomial degree increases the convergence order of the numerical scheme, a p–refinement can reduce the error more effectively, at least in the asymptotic range.

4.1.3. hp–refinement

While h–refinement improves the geometrical solution of the discretization and is thus well suited to capture local characteristics like discontinuities of the solution, p–refinement leads to a higher order of convergence and is accordingly more effective in smooth parts of the solution, if high accuracy is needed. Combining the two procedures using h–refinement in non–smooth parts of the solution and p–refinement in smooth parts, both advantages can be achieved. Applying an adaptive refinement strategy as described in Section 4.2, the decision on one of the basic refinement strategies can be made for each individual element, resulting in an optimal convergence of the overall scheme.

4.1.4. Node movement, r–refinement

The refinement strategies described so far have in common, that they do not change the mesh regularity. The old discretization can always be found as a subset of the new one. It might be useful, however, to change the mesh in a more drastical way. Using a fixed number of elements and a fixed polynomial degree, the error of the approximated solution is still dependent on the position of nodes, which determine the size, position and shape of the mesh elements. Concentrating nodes in regions with higher error and using less elements in regions with less significant flow features the error can be reduced. Note, that this does not increase the asymptotic order of convergence. It is possible, however, to reduce the constant involved in the error estimation and to reach the asymptotic behavior earlier. The name r–refinement is related to the *relocation* of nodes.
In order to decide upon a desirable node distribution it is necessary to evaluate the approximate solution and obtain information on the error distribution. A different approach is to solve an optimization problem, in which the node positions are optimized with respect to the global error. As the actual problem has to be solved in each step of this optimization, this procedure is quite expensive. It could be useful, however, to apply such an r–refinement step to a coarse mesh. Having obtained a better distribution of nodes to start with, the improved mesh can be used for subsequent refinements using h–, p– or hp–refinement strategies.
As node movement completely changes the geometrical discretization of the domain, it is not possible to perform a lossless interpolation of the solution from the old to the new discretization, furthermore the interpolation is more complicated and probably more expensive. This will generally result in a higher number of iterations to converge the solution of the discretized equations. Furthermore, it is very difficult to avoid the

occurrence of irregular elements after moving the nodes, which additionally limits the applicability of such a refinement strategy.

4.1.5. Remeshing

Node movement can be done globally or locally. In both cases the number of nodes and elements is not changed. Using information from the old approximated solution to create a new mesh with an improved distribution of nodes and an increased number of elements results in a complete remeshing. As there are no artificial restrictions on the new mesh, this method offers the best chance of creating an optimal quality mesh for the new discretization. It is, however, not a trivial task to generate the information from which the geometry of this optimal mesh can be created. Several authors use metrics constucted from approximate second derivatives of the approximate solution for this purpose, cf. [6, 7, 18, 19]. Only triangular meshes have been used so far. The newly created mesh is required to have edge lengths of unity with respect to the constructed mesh metric field. Remeshing is an expensive procedure, however, and the construction of meshes based on metric fields using publicly available codes is only possible for two–dimensional domains so far. Concerning an interpolation of the solution, the same restrictions apply as for r–refinement.

Considering implementational issues it is necessary or at least helpful to decide on one type of refinement strategies. The "embedded refinement strategies" h–, p– and hp–refinement allow an optimal interpolation of the solution and can thus accelerate the solution process. Furthermore it is possible to reuse the embedded structure of the refined discretizations for multigrid algorithms. Because of these advantages, the following part of this work will only be concerned with the first type of refinement strategies. In particular, only h–refinement will be used for the numerical examples. However, most of the following techniques can be applied to p–refinement as well. In order to use hp-refinement, an additional criterion to decide upon one of the strategies would be required.

4.2. Adaptive mesh refinement

While a global refinement, in which each element is refined, ensures convergence of the approximated solution, it considerably increases the computational effort. If each edge of the triangulation is split into two new edges, the number of elements will increase by a factor of 2^d. Limited resources will generally allow only few global refinement steps, if any at all. However, in many cases it is sufficient to refine only those parts of the triangulation, where large errors occur. The location of these elements depends on the characteristic features of the actual problem and its solution as well as on the triangulation. As the exact solution is unknown[5], the location of errors has to be evaluated based on the approximate solution $\mathbf{u}_h$ before refinement can take place. As such a refinement locally adapts the triangulation to the actual problem and its solution, the process is known as *adaptive mesh refinement*.

[5] If the solution was known, there would be no need to use a numerical method. Clearly, problems with known solutions are often exploited in order to evaluate the effectiveness of a numerical method, but algorithms for adaptive mesh refinement may not rely on the knowledge of the exact solution.

Having obtained an indication on the location of errors (see the next chapter for details), the elements with the largest error can be flagged for refinement, whereas those with the smallest contribution to the global error can be coarsened in order to save computational resources. In general, this flagging is done according to certain threshold values θ_r for refinement and θ_c for coarsening. An element is flagged for refinement, if the local error estimate η_κ is large enough ($\eta_\kappa \geq \theta_r$), flagged for coarsening if it is small enough ($\eta_\kappa \leq \theta_c$) and left unchanged otherwise. Of course the condition $\theta_c < \theta_r$ has to be fulfilled in order to make the decision unique. There exist different strategies to choose the threshold values. Three distinct possibilities will be presented and discussed briefly.

1. The threshold values can be chosen as fixed values. This is easy from an implementational point of view and enables to flag an element without knowledge of the estimates on other elements. However, it is difficult to choose appropriate values, as the general magnitude of errors has to be known before the approximate solution is obtained, otherwise the adapted mesh would be inadequate for the specific problem. Furthermore, the magnitude of errors decreases in the course of repeated refinements, as the global error tends to zero and this error is distributed to an increasing number of elements. Therefore, this strategy is very difficult and cannot be applied in pratice.

2. The threshold values can be chosen based on an analysis of all local error indicators. Specified fractions f_r and f_c of the elements with the largest and smallest error indicators are to be refined or coarsened, respectively. Having obtained the indicators on all elements, these values have to be ordered by size. The threshold values are then chosen at the appropriate positions of the ordered vector of indicators to ensure, that for $f_r \cdot n_{\text{el}}$ elements the indicator is greater or equal than the threshold value for refinement, whereas it is smaller than the threshold value for coarsening for $f_c \cdot n_{\text{el}}$ elements, where n_{el} is the total number of (active) elements. Of course, the condition $f_c + f_r < 1$ has to be fulfilled for uniqueness. This strategy allows the best control over the number of elements after the refinement step[6]. As this number directly determines the resources needed for obtaining the solution on the refined triangulation, this control makes the presented strategy the best choice in most cases.

   尽管特别记了哪些网格需要细化但也不能完全确定细化后网格的数量

3. Based on the idea of local error indicators that are ordered by size, the threshold values can be calculated in a way which ensures that the elements flagged for refinement contribute a certain fraction g_r to the global error. Adding the elements with largest error first, the number of elements to be refined can be minimized. The same principle can be applied to the coarsening of elements using a respective fraction $g_c < 1 - g_r$. Elements to be coarsened are added starting with the smallest local errors in order to coarsen a maximum number of elements. While this strategy offers the best adaption to the characteristics of the actual solution, it gives only limited control of the number of elements after the refinement. In some cases,

[6] Most numerical codes require the number of *hanging nodes* to be limited, i.e. a face of an element may only be refined a certain number of times. This condition gives rise to the need of refining not only the flagged elements, but additional ones in the vicinity in order to yield a valid mesh after the refinement step. Furthermore, additional elements might be flagged due to mesh smoothing algorithms, see Section A.4 in the appendix. Because of these reasons, the number of elements on the next refinement level cannot be determined directly from the number of flagged elements.

only very little elements contribute largely to the global error and are, accordingly, flagged for refinement. Such a low increase in the number of elements might slow down the overall convergence as measured in refinement steps needed to obtain a certain accuracy of the approximated solution.

In general, only the approximate solution obtained on the finest mesh is of importance, all the coarser calculations are just needed in order to obtain the final triangulation. Refining only very few elements in each refinement step, the number of elements on the final triangulation can be minimized, but the number of cycles needed to obtain that triangulation increases. In order to reduce the overall effort, each refinement step should increase the number of elements considerably. Using the strategy of refining and coarsening a fixed fraction of elements, see 2. above, values of $f_r = 0.2$ and $f_c = 0.1$ are good choices for many problems.
However, choosing the appropriate threshold values is only of minor importance as compared to the problem of obtaining accurate estimates of the local error. This problem will be treated in Chapter 5.

5. Error estimation

In order to drive an adaptive refinement strategy, two aspects of the error have to be estimated. As a first aspect, information on the global error is needed in order to determine the overall accuracy of the approximated solution. This information is needed to evaluate a stopping criterion. Alternatively, the adaptive process could also be stopped after a fixed number of refinement steps or when a certain number of elements is reached. However, in general the calculation is needed for a certain purpose, and this purpose requires a specific accuracy. Only an evaluation of the global error can ensure, that this accuray is (approximately) achieved and that the refinement cycle is stopped at that point in order not to waste computational resources on improving the solution to unneeded accuracy.
The second and equally important aspect is the estimation of local errors, as these are the key component driving an individual local refinement step. Concerning local estimates it is generally not important to achieve a very good estimate on the exact error. It is more important, that all the estimates are of equal quality. If a larger estimated error corresponds to a larger exact error, the adaptive refinement process will refine the cells, that would be refined using the exact error for evaluation and thus be effective.

局部误差的估计不需要准但需要对所有的格"视同仁"

There are two general classes of local error indicators. 1 One class is based on general features of the approximated solution. These features can include gradients or higher order derivatives. They give indications, where large errors can occur, but they do not tend to zero if the exact error does due to mesh refinement. Furthermore, the correlation between the exact error and these indicators can be poor, depending on the actual problem to be solved. 2 The indicators belonging to the second class are usually more involved, but they deliver estimates of the magnitude of the error. If these estimates are appropriate, they tend to zero in the same way as the exact error does. In addition to a better representation of local errors, these indicators allow to sum up local contributions in order to obtain an estimate on the global error. Though such indicators are generally more costly than simple feature-based ones, the additional effort is balanced in most cases by more accurately refined triangulations. Therefore, only the second class of error indicators will be considered here.

5.1. Hierarchical error estimation

A very simple way of estimating the error is based on hierarchical approximations. In the case of h–refinement a second approximation $\mathbf{u}_{h/2}$ to the solution is obtained on a globally refined triangulation. Using the fact, that the error tends to zero in the case of global refinement it can be assumed, that the error $\mathbf{e}^{(h/2)}$ on the refined triangulation, measured in an appropriate norm, is small compared to the error $\mathbf{e}^{(h)}$ on the original triangulation: $\|\mathbf{e}^{(h/2)}\| \ll \|\mathbf{e}^{(h)}\|$. This way the error can be approximated by

$$\mathbf{u}_{h/2} - \mathbf{u}_h = \left(\mathbf{u} - \mathbf{e}^{(h/2)}\right) - \left(\mathbf{u} - \mathbf{e}^{(h)}\right) = \mathbf{e}^{(h)} - \mathbf{e}^{(h/2)} \approx \mathbf{e}^{(h)}. \tag{5.1}$$

This procedure allows to evaluate the global error as well as local contributions. However, solving the problem on a globally refined mesh is too expensive in most cases. The adaptive refinement process is introduced in order to avoid using global refinements. The effort of solving the problem on a globally refined triangulation only to evaluate the elements with large errors seems disproportionate. The solution obtained in this intermediate global refinement is more accurate than that on the next adaptively refined triangulation. Though this procedure can still help saving cells in the course of several refinement steps it should not be considered a useful option[7]. Other error indicators can achieve reasonable estimates with much less effort.

5.2. Adjoint based indicators

Obtaining the solution of a flow field is in many applications only the necessary step on the way to evaluate certain target functionals $J(\mathbf{u})$, which depend on the solution. In aerodynamic applications important target functionals are force and momentum coefficients. An accurate solution of the flow is only needed in order to achieve high accuracy in these quantities. An error indicator that measures the error of these quantities instead of the flow error itself is desirable. Such indicators can drive a *goal oriented* adaptive refinement, in which the triangulation is only refined in those regions that have errors in the flow variables which contribute largely to the error of the selected target functional. Consider, for example, an airfoil in a supersonic flow. Whereas it is necessary to resolve the whole bow shock in front of the leading edge in order to obtain an accurate solution of the flow field, it has been shown in [13, 10] that it is sufficient to resolve this shock in the vicinity of the airfoil, if only the force coefficients are to be evaluated. To be more precise, refining some regions with smaller changes in the flow, which are in the vicinity and upstream of the airfoil, might prove more effective than resolving the bow shock far from the leading edge. While such general considerations help understanding the principle of goal oriented adaptive refinement, an accurate method to identify the importance of certain flow regions for the evaluation of target functionals is needed to enable such a procedure to work in general. The solution of an *adjoint problem* or *dual problem* provides this information. Dual-weighted residuals represent reliable indicators of the error in terms of target functionals.

Target functionals

The following target functionals are useful for aerodynamic applications:

- Pressure induced drag and lift coefficients: c_{dp} and c_{lp}

$$J(\mathbf{u}) = \frac{1}{q_\infty \, \bar{l}} \int_{\Gamma_W} p \, \mathbf{n} \cdot \psi \, \mathrm{d}s, \tag{5.2}$$

[7] The additional effort of solving the problem on a globally refined triangulation may be affordable, if the obtained refined mesh is used for several flow calculations afterwards. This can be the case in the numerical approximation of transient problems, if the obtained mesh is used for several time steps. A similar strategy is described by Sun and Wheeler [20], who use global anisotropic refinement in order to decide upon anisotropic features of the solution.

- Viscous drag and lift coefficients: c_{df} and c_{lf}

$$J(\mathbf{u}) = -\frac{1}{q_\infty \bar{l}} \int_{\Gamma_W} (\underline{\tau}\,\mathbf{n}) \cdot \psi \,\mathrm{d}s, \tag{5.3}$$

- Total drag and lift coefficients: $c_{\mathrm{d}} = c_{\mathrm{dp}} + c_{\mathrm{df}}$ and $c_{\mathrm{l}} = c_{\mathrm{lp}} + c_{\mathrm{lf}}$

$$J(\mathbf{u}) = \frac{1}{q_\infty \bar{l}} \int_{\Gamma_W} (p\,\mathbf{n} - \underline{\tau}\,\mathbf{n}) \cdot \psi \,\mathrm{d}s, \tag{5.4}$$

where $q_\infty = \frac{1}{2}\gamma p_\infty M_\infty^2 = \frac{1}{2}\gamma \frac{|\mathbf{v}_\infty|^2}{c_\infty^2} p_\infty = \frac{1}{2}\rho_\infty |\mathbf{v}_\infty|^2$ is the freestream dynamic pressure, M denotes the Mach number, c the speed of sound defined by $c^2 = \gamma p/\rho$ and $\bar{l}$ denotes a reference length. The subscripts $_\infty$ indicate freestream quantities. ψ is given by

$$\psi_d = (\cos(\alpha), \sin(\alpha))^\top \tag{5.5}$$

or

$$\psi_l = (-\sin(\alpha), \cos(\alpha))^\top \tag{5.6}$$

for the drag and lift coefficient, respectively.

The adjoint problem

带权重的误差估计

In order to obtain weighted error indicators, the adjoint problem has to be solved. This procedure will be briefly introduced here, following the description of Hartmann [9].
If the target functional $J(\cdot)$ is differentiable, which will be assumed in the following, the mean-value linearization $\bar{J}(\cdot,\cdot;\cdot)$ of $J(\cdot)$ is defined by

$$\bar{J}(\mathbf{u}, \mathbf{u}_h; \mathbf{u} - \mathbf{u}_h) = J(\mathbf{u}) - J(\mathbf{u}_h) = \int_0^1 J'[\theta\mathbf{u} + (1 - \theta\mathbf{u}_h](\mathbf{u} - \mathbf{u}_h)\,\mathrm{d}\theta, \tag{5.7}$$

where $J'[\mathbf{w}](\cdot)$ denotes the Fréchet derivative of $J(\cdot)$ evaluated at $\mathbf{w}$. Obviously, the term $J(\mathbf{u}) - J(\mathbf{u}_h)$ represents the error in the target functional, if this funtional is evaluated using the approximate solution $\mathbf{u}_h$ instead of the exact solution $\mathbf{u}$.
Similarly, the mean-value linearization $\mathcal{M}(\cdot,\cdot;\cdot,\cdot)$ of a nonlinear operator $\mathcal{N}(\cdot,\cdot)$ [8] is given by

$$\mathcal{M}(\mathbf{u}, \mathbf{u}_h; \mathbf{u} - \mathbf{u}_h, \mathbf{v}) = \mathcal{N}(\mathbf{u}, \mathbf{v}) - \mathcal{N}(\mathbf{u}_h, \mathbf{v}) = \int_0^1 \mathcal{N}'_{\mathbf{u}}[\theta\mathbf{u} + (1-\theta)\mathbf{u}_h](\mathbf{u} - \mathbf{u}_h, \mathbf{v})\,\mathrm{d}\theta \tag{5.8}$$

for all $\mathbf{v}$ in $\mathbf{V}$, where $\mathcal{N}'_{\mathbf{u}}[w](\cdot, \mathbf{v})$ denotes the Fréchet derivative of $\mathbf{u} \mapsto \mathcal{N}(\mathbf{u}, \mathbf{v})$.
Using this notation, the dual or adjoint problem is given by: find $\mathbf{z} \in \mathbf{V}$ such that

$$\mathcal{M}(\mathbf{u}, \mathbf{u}_h; \mathbf{w}, \mathbf{z}) = \bar{J}(\mathbf{u}, \mathbf{u}_h; \mathbf{w}) \quad \forall \mathbf{w} \in \mathbf{V}. \tag{5.9}$$

In the following it is assumed, that problem (5.9) is well-posed, i.e. that is has a unique solution. Generally this depends on both the target functional and the definition of $\mathcal{M}$. The specific target functionals given above result in well-posed problems.

[8] See Equation (3.10) in Section 3.4 for the definition of the nonlinear operator $\mathcal{N}$ for the Navier–Stokes equations.

The exact solution $\mathbf{z}$ is generally unknown and has to be approximated numerically. However, as Equation (5.9) involves the unknown solution $\mathbf{u}$ to the primal problem, the exact adjoint problem cannot be treated numerically. Therefore, problem (5.9) is linearized, the linearizations involved in equations (5.7) and (5.8) are performed about the approximated solution $\mathbf{u}_h$. This leads to the *linearized adjoint problem*: find $\hat{\mathbf{z}} \in \mathbf{V}$ such that

$$\mathcal{N}'_{\mathbf{u}}[\mathbf{u}_h](\mathbf{w}, \hat{\mathbf{z}}) = J'[\mathbf{u}_h](\mathbf{w}) \quad \forall \mathbf{w} \in \mathbf{V}, \tag{5.10}$$

In order to approximate (5.10) numerically, $\hat{\mathbf{z}}$ and $\mathbf{w} \in \mathbf{V}$ are replaced by discrete functions $\hat{\mathbf{z}}_h$ and $\mathbf{w}_h \in \mathbf{V}_h^{\hat{p}}$, respectively. Thereby the approximate adjoint problem is given by: find $\hat{\mathbf{z}}_h \in \mathbf{V}_h^{\hat{p}}$ such that

$$\mathcal{N}'_{\mathbf{u}}[\mathbf{u}_h](\mathbf{w}_h, \hat{\mathbf{z}}_h) = J'[\mathbf{u}_h](\mathbf{w}_h) \quad \forall \mathbf{w}_h \in \mathbf{V}_h^{\hat{p}}. \tag{5.11}$$

As $\hat{p}$ might be different from p, the derivative $\mathcal{N}'_{\mathbf{u}}$ of the nonlinear operator has to be assembled with respect to $\mathbf{V}_h^{\hat{p}}$ rather than $\mathbf{V}_h^p$.
The above outline represents the *discrete adjoint approach*, as it is based on the linearization of the discrete primal equations. Obtaining the adjoint equations of the continuous primal problem and performing an discretization afterwards represents the *continuous adjoint approach* instead.

Error estimation based on the adjoint problem

Assuming the knowledge of the exact solution $\mathbf{z}$ to the dual problem (5.9), using the consistency

$$\mathcal{N}(\mathbf{u}, \mathbf{v}) = 0 \quad \forall \mathbf{v} \in \mathbf{V}, \tag{5.12}$$

as well as the Galerkin orthogonality of the discretization

$$\mathcal{N}(\mathbf{u}, \mathbf{v}_h) - \mathcal{N}(\mathbf{u}_h, \mathbf{v}_h) = 0 \quad \forall \mathbf{v}_h \in \mathbf{V}_h^p, \tag{5.13}$$

and choosing $\mathbf{w} = \mathbf{u} - \mathbf{u}_h$, the error in the target functional can be written as

$$\begin{aligned} J(\mathbf{u}) - J(\mathbf{u}_h) &= \bar{J}(\mathbf{u}, \mathbf{u}_h; \mathbf{u} - \mathbf{u}_h) = \mathcal{M}(\mathbf{u}, \mathbf{u}_h; \mathbf{u} - \mathbf{u}_h, \mathbf{z}) \\ &= \mathcal{N}(\mathbf{u}, \mathbf{z}) - \mathcal{N}(\mathbf{u}_h, \mathbf{z}) \\ &= \mathcal{N}(\mathbf{u}, \mathbf{z}) - \mathcal{N}(\mathbf{u}_h, \mathbf{z}) - [\mathcal{N}(\mathbf{u}, \mathbf{z}_h) - \mathcal{N}(\mathbf{u}_h, \mathbf{z}_h)] \\ &= \mathcal{N}(\mathbf{u}, \mathbf{z} - \mathbf{z}_h) - \mathcal{N}(\mathbf{u}_h, \mathbf{z} - \mathbf{z}_h) \\ &= -\mathcal{N}(\mathbf{u}_h, \mathbf{z} - \mathbf{z}_h) \end{aligned} \tag{5.14}$$

for all $\mathbf{z}_h$ in $\mathbf{V}_h^p$. This way the exact error in the target functional can be evaluated without knowledge of the exact solution $\mathbf{u}$ of the primal problem. Equation (5.14) involves the exact solution $\mathbf{z}$ to the dual problem however, which is also unknown in general. Therefore $\mathbf{z}$ has to be approximated, resulting in an approximate error representation:

$$J(\mathbf{u}) - J(\mathbf{u}_h) \approx -\mathcal{N}(\mathbf{u}_h, \hat{\mathbf{z}}_h - \mathbf{z}_h), \tag{5.15}$$

where $\hat{\mathbf{z}}_h$ is the approximate solution to the linearized adjoint problem (5.10) according to Equation (5.11) and $\mathbf{z}_h = 0$ can be chosen for convenience. It has to be noted, that any $\mathbf{z}_h$ in $\mathbf{V}_h^p$ can be chosen in Equation (5.14), this includes choosing $\mathbf{z}_h = 0$ as above. However, it also allows to choose $\mathbf{z}_h = \hat{\mathbf{z}}_h$, if $\hat{\mathbf{z}}_h \in \mathbf{V}_h^p$, i.e. if $\hat{p} = p$. As this leads to

$\hat{\mathbf{z}}_h - \mathbf{z}_h = 0$, the approximate error representation according to Equation (5.15) would evaluate to zero:

$$J(\mathbf{u}) - J(\mathbf{u}_h) \approx \mathcal{N}(\mathbf{u}_h, \hat{\mathbf{z}}_h - \mathbf{z}_h) = \mathcal{N}(\mathbf{u}_h, 0) = 0 \quad \forall \hat{\mathbf{z}}_h \in \mathbf{V}_h^p. \tag{5.16}$$

In order to obtain useful results, the linearized adjoint problem has to be solved on the same mesh $\mathcal{T}_h$, but with a higher polynomial degree $\hat{p} > p$, usually $\hat{p} = p + 1$ gives good results.
Equation (5.14) can be rewritten to include local information in form of a sum of local errors over all cells κ:

$$J(\mathbf{u}) - J(\mathbf{u}_h) = -\mathcal{N}(\mathbf{u}_h, \mathbf{z} - \mathbf{z}_h) = \sum_{\kappa \in \mathcal{T}_h} \eta_\kappa, \tag{5.17}$$

where the local errors η_κ are the local cell-wise contributions to the error measured in terms of the target functional $J(\cdot)$. An explicit expression of the weighted residual indicators η_κ is given in [15].
Applying the triangle inequality to Equation (5.17) yields the following upper bound on the target functional error:

$$|J(\mathbf{u}) - J(\mathbf{u}_h)| \leq \sum_{\kappa \in \mathcal{T}_h} |\eta_\kappa|. \tag{5.18}$$

The same decomposition into local, cell-wise terms can be used when $\mathbf{z}$ is replaced by an approximate $\hat{\mathbf{z}}_h$. This results in error representations and bounds which are approximate only. Numerical experiments show, however, that they are still reliable and useful for practical computations, cf. [15].

5.3. Residual based indicators

The error indicators presented in Section 5.2 are based on weighted residuals of the approximate solution. The weighting is done using the (approximate) solution of an adjoint problem and has the physical meaning of choosing those flow regions with large contributions to the error of a certain target functional. Obtaining the solution of this adjoint problem requires an additional computational effort, therefore it might be desirable to derive indicators, which do not depend on the adjoint solution. Assuming that $\mathbf{z} \in [H^s(\Omega)]^4, 2 \leq s \leq p + 1$ and that a constant C_z can be found, such that $\|\mathbf{z}\|_{H^s(\omega)} \leq C_z$, according to Hartmann and Houston [15] an upper bound of the target functional error is given by

$$|J(\mathbf{u}) - J(\mathbf{u}_h)| \leq C \left(\sum_{\kappa \in \mathcal{T}_h} (\eta^{\text{res}})^2 \right)^{\frac{1}{2}} \tag{5.19}$$

with a constant C that is independent of the mesh size h.

The local residual-based indicators η^{res} are given by

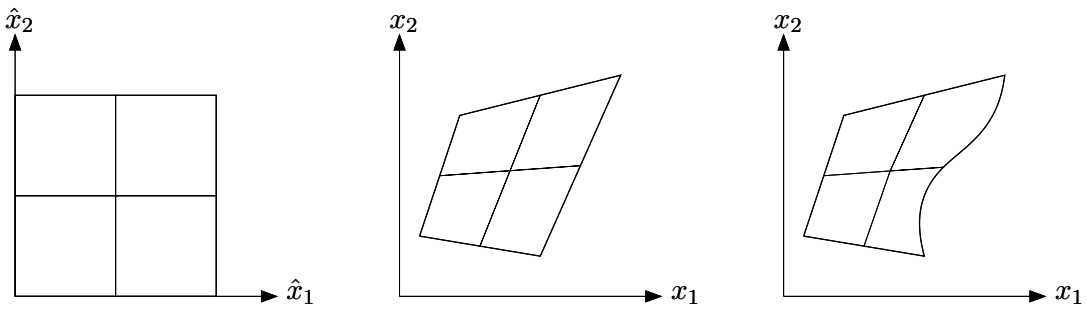
$$\begin{aligned}
\eta_\kappa^{\text{res}} &= \|h_\kappa^s \mathbf{R}(\mathbf{u}_h)\|_{L_2(\kappa)} + \|h_\kappa^{s-1/2}(\mathcal{F}^c(\mathbf{u}_h)\cdot\mathbf{n}_\kappa - \mathcal{H}(\mathbf{u}_h^+,\mathbf{u}_h^-,\mathbf{n}_\kappa))\|_{L_2(\partial\kappa\setminus\Gamma)} \\
&+ \|h_\kappa^{s-1/2}(\mathcal{F}^c(\mathbf{u}_h)\cdot\mathbf{n}_\kappa - \mathcal{H}_\Gamma(\mathbf{u}_h^+,\mathbf{u}_\Gamma(\mathbf{u}_h^+),\mathbf{n}_\kappa))\|_{L_2(\partial\kappa\cap\Gamma)} \\
&+ \|h_\kappa^{s-3/2} G_{\cdot j}\underline{[\![\mathbf{u}_h]\!]}_j\|_{L_2(\partial\kappa\setminus\Gamma)} + \|h_\kappa^{s-1/2}[\![\mathcal{F}^v(\mathbf{u}_h,\nabla\mathbf{u}_h)]\!]\|_{L_2(\partial\kappa\setminus\Gamma)} \\
&+ \|h_\kappa^{s-1/2}\delta\left(\mathbf{u}_h^+ - \mathbf{u}_h^-\right)\|_{L_2(\partial\kappa\setminus\Gamma)} + \|h_\kappa^{s-1/2}\delta\left(\mathbf{u}_h^+ - \mathbf{u}_\Gamma(\mathbf{u}_h^+)\right)\|_{L_2(\partial\kappa\cap\Gamma)} \\
&+ \|h_\kappa^{s-1/2}\left(\mathcal{F}^v(\mathbf{u}_h^+,\nabla\mathbf{u}_h^+) - \mathcal{F}_\Gamma^v(\mathbf{u}_h^+,\nabla\mathbf{u}_h^+)\right)\cdot\mathbf{n}_\kappa\|_{L_2(\partial\kappa\cap\Gamma)} \\
&+ \|h_\kappa^{s-3/2} G_{\cdot j}(\mathbf{u}_h^+)\left[\left(\mathbf{u}_h^+ - \mathbf{u}_\Gamma(\mathbf{u}_h^+)\right)\otimes\mathbf{n}\right]_j\|_{L_2(\partial\kappa\cap\Gamma)},
\end{aligned} \tag{5.20}$$

where $\mathbf{R}(\mathbf{u}_h)|_\kappa = -\nabla\cdot\mathcal{F}^c(\mathbf{u}_h) + \nabla\cdot\mathcal{F}^v(\mathbf{u}_h,\nabla\mathbf{u}_h)$, $\kappa\in\mathcal{T}_h$. This indicator has been derived in [15], where here the indicators are adjusted according to the adjoint consistent boundary discretization, see [11, 12]. The adjusted form is taken from [16].

6. Anisotropic refinement

6.1. Definition of isotropic and anisotropic refinement

Isotropic refinement Using an isotropic refinement strategy, the edges of the reference element (unit square for $d = 2$, unit cube for $d = 3$) are split exactly in half, resulting in 2^d new elements. Considering the unit square in two dimensions, Figure 6.1(a) gives a graphical representation of that process. For elements bounded only by straight edges, the same applies for the element in real space coordinates, as shown in Figure 6.1(b). For elements on the boundary with a curved boundary edge, the new vertex on the boundary edge may not be exactly in the middle of the edge. Furthermore, the position of the middle vertex is not obvious and has to be defined, usually by means of a weighted mean value of both the old corner vertices and the new vertices on the edges. Applying a large enough weight on the new vertices helps to avoid badly shaped quadrilaterals on the boundary. Figure 6.1(c) shows an example of the refinement of an element with curved edges. It has to be noted, that generally the curved boundary is approximated by a piecewise polynomial of fixed degree. Using more pieces after the refinement, the boundary representation can change slightly. However, for the sake of simplicity this effect is not shown in Figure 6.1(c).



(a) reference element

(b) element in real space with straight boundaries

(c) element in real space with a curved boundary

Figure 6.1.: Isotropic refinement for quadrilateral elements.

Considering three dimensions, the extension from the two–dimensional case is quite straight-forward. In addition to new vertices on edges and in the middle of the element there are now also new vertices on the middle of the faces bounding the element. This has to be considered on curved boundaries. For simplicity, Figure 6.2 visualizes the isotropic refinement process in three dimensions only on the reference element.

Anisotropic refinement The anisotropic refinement considered in this work is based on splitting only a part of the edges in half. In order to result in quadrilaterals or hexahedra,

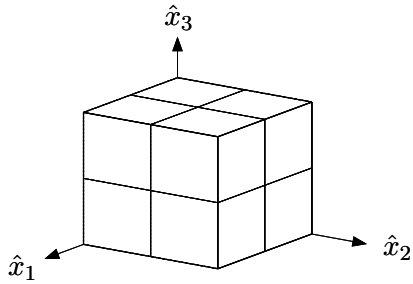


Figure 6.2.: Isotropic refinement for the hexahedral reference element.

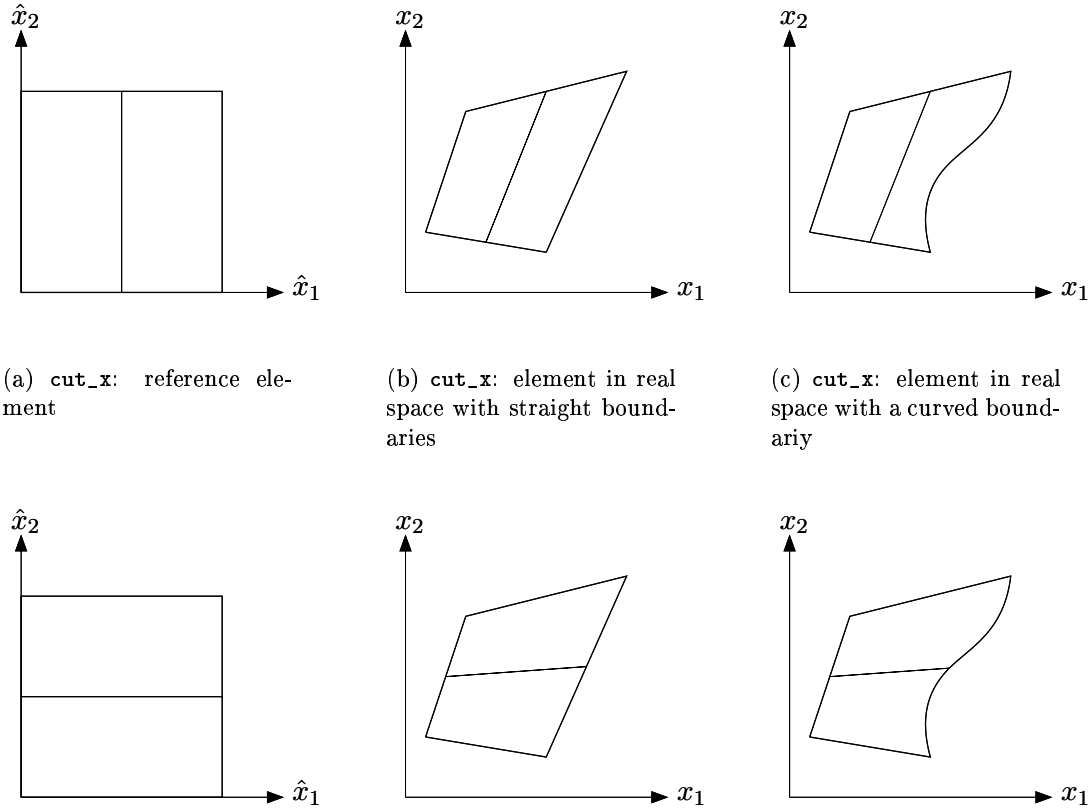
respectively, for the children as well, whole sets of parallel edges on the reference element have to be either split or stay unchanged. Out of the d distinct sets of parallel edges, $t < d$ sets are split, resulting in 2^t new elements, i.e. two children elements in two dimensions and two or four children in three dimensions. Additionally allowing $t = d$ as limit value, anisotropic refinement can be considered to be a generalization of isotropic refinement. Formally it is also possible, to consider isotropic refinement as a special case of anisotropic refinement. The naming of anisotropic refinement cases follows this formal notion.
As a convention, anisotropic refinement cases are named acoording to the axes, along which the edges are split in half. Table 6.1 gives an overview of the possible refinement cases.

refinement case	refined along $\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	number of children	relevant for 2D	3D
`cut_x`	✓			2	✓	✓
`cut_y`		✓		2	✓	✓
`cut_xy`	✓	✓		4	✓	✓
`cut_z`			✓	2		✓
`cut_xz`	✓		✓	4		✓
`cut_yz`		✓	✓	4		✓
`cut_xyz`	✓	✓	✓	8		✓

Table 6.1.: Anisotropic refinement cases (on the reference element).

Clearly, only the first three cases are relevant for two–dimensional domains, as in these there is no third coordinate axis. Furthermore, the refinement case `cut_xy` recovers isotropic refinement in two dimensions, whereas `cut_xyz` is a synonym for isotropic refinement in three dimensions. As indicated in Section 3.2, curved edges are only considered at the boundary of the domain. Therefore, in the case of anisotropic refinement with $t < d$ no vertex in the middle of the element has to be calculated, eliminating the need to define a weighting rule. The new vertices in the middle of split edges are always connected by straight lines[9]. Figure 6.3 visualizes the anisotropic refinement cases `cut_x` and `cut_y` for the two dimensional case for the reference element as well as for elements in real space. The isotropic refinement case `cut_xy` has been shown in Figure 6.1. It has

[9]In the case of strongly curved boundaries this may lead to strangely shaped elements. If the curvature is strong enough, it is even possible, that the edge on the boundary intersects the opposite edge. In order to avoid such cases, isotropic refinement has to be enforced, if the boundary is too far inside the element.



(a) `cut_x`: reference element

(b) `cut_x`: element in real space with straight boundaries

(c) `cut_x`: element in real space with a curved boundariy

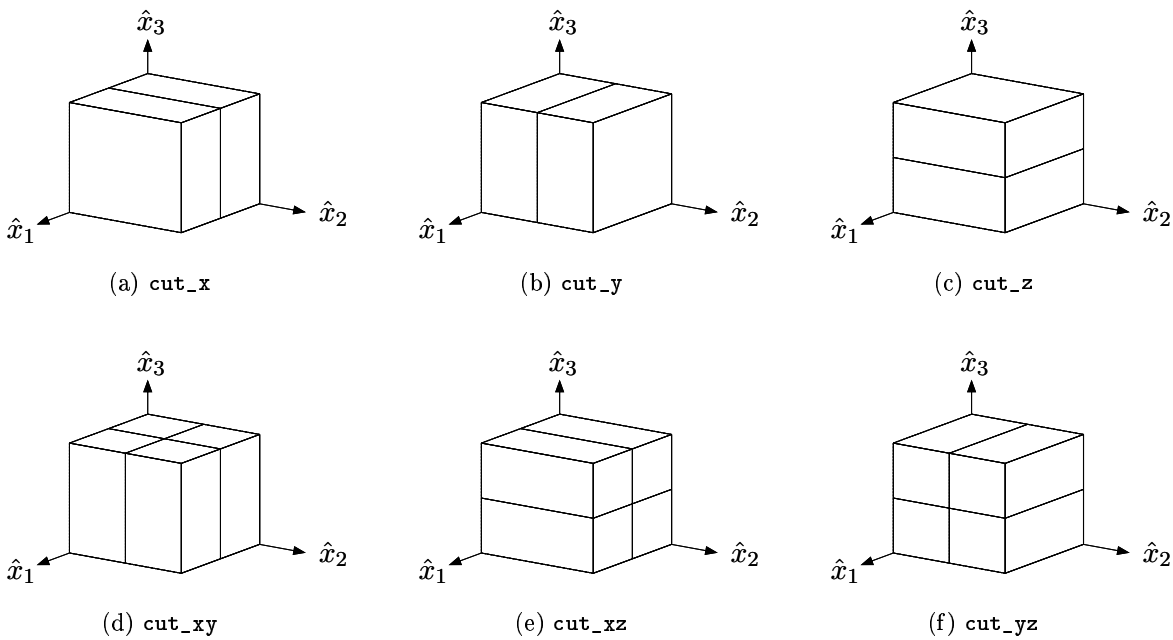
(d) `cut_y`: reference element

(e) `cut_y`: element in real space with straight boundaries

(f) `cut_y`: element in real space with a curved boundary

Figure 6.3.: Anisotropic refinement for quadrilateral elements, refinement cases `cut_x` and `cut_y`.

to be noted, that the mapping between reference element and real space element depends on the ordering of corner vertices. Thus it is possible, that the refinement case `cut_x` on the reference element corresponds to the case shown in Figure 6.3(b) or that shown in Figure 6.3(e), depending on the mapping. Because of that, any decision concerning a refinement case has to consider the mapping, as the actual refinement is done using the reference element.
The extension to three dimensions is once again obvious. Like above, only the refinement cases for the reference element will be shown here. Figure 6.4 visualizes the truely anisotropic refinement cases `cut_x`, `cut_y`, `cut_z`, `cut_xy`, `cut_xz` and `cut_yz`, whereas the isotropic case `cut_xyz` can be seen in Figure 6.2.



(a) `cut_x` (b) `cut_y` (c) `cut_z`

(d) `cut_xy` (e) `cut_xz` (f) `cut_yz`

Figure 6.4.: Anisotropic refinement cases for the hexahedral reference element.

6.2. Anisotropic refinement indicators

The error indicators described in Chapter 5 are well suited for controlling the adaptive refinement process in case of isotropic refinement, as they deliver information about *where* large errors occur, i.e. which elements should be refined. They do not deliver any information about the *direction* connected with these errors, which would be needed for information on anisotropic behavior. Unfortunately, the indicators described so far cannot easily be extended to deliver directional information.
There are error indicators however, which include directional information or can be extended by simple means to do this. Such indicators are based on features of the solution like gradients or higher order derivatives. Generally, such feature-based indicators are not very reliable in predicting the elements with the largest error, which makes them a poor choice in most cases. On the other hand, the information about which elements ought to be refined can be obtained by an elaborate error indicator. In a second step,

these elements can be analyzed in order to obtain information on directional behavior. This second aspect can be done using feature-based indicators.
The indicators presented in this work are based on this combined approach of using feature-based directional information on elements which are pre-selected using a highly reliable error estimation technique.
The indicators described in the following sections use two distinct ideas. On the one hand, jumps of the discontinuous solution can be analyzed in order to obtain knowledge of the anisotropic features of the solution, on the other hand this information can be derived using approximations of derivatives of the solution.

6.2.1. Jump indicator

The most characteristic feature of DG methods is the ability to obtain discontinuous approximations. The solution can have jumps over the faces of the triangulation, whereas it is smooth inside each element. These jumps allow some flexibility in approximating the local properties of the solution. With successive mesh refinement these jumps tend to zero, as the real solution is approximated with less error.
Based on this observation it seems justified to assume, that a large jump indicates a larger error as compared to a smaller jump. In fact, integrating the jump over all faces of an element results in an error indicator that can be quite useful for DG(0) methods.
In view of an anisotropic evaluation a large jump over one face indicates, that the mesh size perpendicular to this face is too coarse to sufficiently resolve the solution. In this sense jumps can be used to derive an anisotropic indicator, that uses information which is specific to the numerical method used to solve the problem.
In order to obtain directional information, the average jump K_i of a function ϕ over the two opposite faces f_i^j, $j = 1, \ldots, d$, perpendicular to one coordinate direction i on the reference element can be evaluated as

$$K_i = \left| \frac{\sum_j \int_{f_i^j} \lfloor \phi \rfloor \, \mathrm{d}s}{\sum_j \operatorname{meas}(f_i^j)} \right|, \quad i, j = 1, \ldots, d, \tag{6.1}$$

where $\lfloor \phi \rfloor = \phi^+ - \phi^-$ denotes the jump of a scalar function ϕ and $\int \cdot \, \mathrm{d}s$ indicates a line integral in two dimensions and a surface integral in three dimensions.
For two dimensions, Equation (6.1) provides two distinct values for each element. If there is a k, $1 \leq k \leq d$, such that the average jump in the direction k is larger than the other one in direction l (or larger than all the other ones in case of higher dimensions) by a certain threshold factor θ,

$$K_k > \theta \max_{\substack{l \neq k \\ 1 \leq l \leq d}} K_l, \tag{6.2}$$

then the element can be marked for anisotropic refinement in the corresponding direction k. If the solution function is vector valued, as is the case for the flow equations, the jump of a scalar function ϕ in Equation (6.1) has to be replaced by an appropriate norm of the vector of jumps, for example the l_2-norm.
The empirical threshold factor θ has to be chosen large enough to ensure that only those elements are flagged for anisotropic refinement, which are located near to strong anisotropic features, otherwise the error would not be reduced sufficiently. On the other hand, however, a smaller value of θ allows more elements to be treated anisotropically,

thereby leading to a decreasing number of total elements. Chapter 7 presents some numerical examples obtained using different values of θ.

6.2.2. Derivative indicators

Interpolation error

Assuming that the numerical method used yields good approximations of the solution, an important contribution to the error is given by the interpolation error, representing the difference between the local solution and the local polynomial approximation obtained by interpolating the solution on the discrete polynomial space. Generally, the solution cannot be described exactly by any polynomial of given degree p, even if the polynomial is an interpolation of the exact solution at the support points.
In DG methods usually the projection is considered instead of interpolation. Given a projection operator Π_p, for example the L^2–projector, projecting the solution to a space of piecewise polynomials of degree p and assuming that the solution ϕ is (weakly) differentiable at least $(p+1)$ times, resulting in the tensor $D^{p+1}\phi$ of $(p+1)$th order derivatives, the projection error is bounded from above by

$$\|\phi - \Pi_p\phi\|_{L_2(\kappa)} \leq C\, h^{p+1} \|D^{p+1}\phi\|_{L_2(\kappa)}, \tag{6.3}$$

with a problem dependent constant C, that is independent of the element size h. A large derivative can be balanced by a small mesh size to result in a small error.
The above form is generally introduced for isotropic meshes. Georgoulis [8] gives an analogous anisotropic error estimate for rectangular, axis-parallel elements with edge length h_1 and h_2 along the x_1 and x_2 axis, respectively:

$$\|\phi - \Pi_p\phi\|_{L_2(\kappa)} \leq C \sum_{i=1}^{d} h_i^{p+1} \|\partial_i^{p+1}\phi\|_{L_2(\kappa)}. \tag{6.4}$$

In order to reduce projection errors, the mesh size has to be small along the direction, in which the derivative is large, whereas it can be moderately high in directions with smaller derivatives. This fact will be used later on to decide upon the direction of a possible anisotropic refinement. Before that can be done, some basic procedures have to be provided, including the approximation of derivatives of discrete functions as well as transformations of derivatives between the reference element and real space coordinates.

Finite Difference approximation of derivatives

For a one-dimensional polynomial of degree p the $(p+1)$th derivative vanishes. In case of a tensor product polynomial of degree p this is true for the main diagonal components of the derivative tensor. Therefore the derivative tensor cannot be calculated locally from the piecewise polynomial approximation, but has to be approximated using additional information from the surrounding elements. On each element the pth order derivative can be approximated using the piecewise polynomial. Setting up a finite difference scheme on an element and its neighbors, the $(p+1)$th order derivative can be approximated. This results in one constant approximation for the derivative tensor at the element's midpoint, which is assumed to be representative for the whole element. It has to be kept in mind, however, that the described evaluation will result in directional information only, whereas

the magnitude of the error does not have to be estimated. Therefore, using a constant approximation seems reasonable.
The way the Finite Difference approximations can be computed on element κ will be described for the gradient, but higher order derivatives can be obtained using the same principle. This description follows the way the derivatives are approximated in `deal.II` [2]. If κ' is a neighboring element and $\mathbf{y}_{\kappa'} = \mathbf{x}_{\kappa'} - \mathbf{x}_\kappa$ describes the distance vector between the centers of the two elements, then $\frac{\phi_h(\mathbf{x}_{\kappa'})-\phi_h(\mathbf{x}_\kappa)}{\|\mathbf{y}_{\kappa'}\|}$ is an approximation of the directional derivative $\nabla\phi(\mathbf{x}_\kappa) \cdot \frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|}$. Multiplying both terms by $\frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|}$ from the left and summing over all neighbors κ', the approximation

$$\sum_{\kappa'} \left(\frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \otimes \frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \right) \nabla\phi(\mathbf{x}_\kappa) \approx \sum_{\kappa'} \left(\frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \frac{\phi_h(\mathbf{x}_{\kappa'}) - \phi_h(\mathbf{x}_\kappa)}{\|\mathbf{y}_{\kappa'}\|} \right) \tag{6.5}$$

is obtained, where $\otimes$ denotes the outer product of two vectors as introduced in Section 3.4. If the matrix $Y = \sum_{\kappa'} \left(\frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \otimes \frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \right)$ is regular, which is the case when the vectors $\mathbf{y}_{\kappa'}$ to all neighbors span the whole space, the gradient can be approximated by

$$\nabla\phi(\mathbf{x}_\kappa) \approx Y^{-1} \sum_{\kappa'} \left(\frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \frac{\phi_h(\mathbf{x}_{\kappa'}) - \phi_h(\mathbf{x}_\kappa)}{\|\mathbf{y}_{\kappa'}\|} \right). \tag{6.6}$$

Approximations to higher derivatives can be computed in a similar way. For example, the tensor of second derivatives is approximated by the formula

$$\nabla^2\phi(\mathbf{x}_\kappa) \approx Y^{-1} \sum_{\kappa'} \left(\frac{\mathbf{y}_{\kappa'}}{\|\mathbf{y}_{\kappa'}\|} \otimes \frac{\nabla\phi_h(\mathbf{x}_{\kappa'}) - \nabla\phi_h(\mathbf{x}_\kappa)}{\|\mathbf{y}_{\kappa'}\|} \right). \tag{6.7}$$

It has to be noted, that unlike the true tensor of second derivatives, its approximation is not necessarily symmetric. This is due to the fact that in the derivation, it is not clear whether the term $\nabla^2\phi\mathbf{y}_{\kappa'}$ or $\mathbf{y}_{\kappa'}^T\nabla^2\phi$ should be considered as projected second derivative. Depending on this choice, one approximation of the tensor of second derivatives or its transpose is obtained. To avoid this ambiguity, the approximation can be symmetrized by calculating the mean value of the approximation and its transpose.
Approximations of higher derivatives can be obtained along the same line and will not be described here.
So far, nothing has been said about the origin of the approximations to the pth order derivative at the element centers, which are needed in order to approximate the $(p+1)$th order derivative. As this approximation is a useful information only in case of piecewise polynomials of degree p (or higher), these quantities can be evaluated on each element using only the local polynomial approximation. This is a comparatively cheap evaluation and yields *local* estimates of the $(p+1)$th order derivative tensor, which are desired in view of refinement information.
A different approach would be to define the derivatives recursively using the above equations. This is much more expensive and would additionally lead to an increasing patch of elements κ' which have influence on the approximation on element κ for higher order derivative approximations. As higher order polynomial approximations generally use a coarser geometric resolution, the effect of obtaining estimates with *global* instead of *local* character is even aggravated. Therefore, using a recursive evaluation should not be considered reasonable.

Transformation to the reference element

In order to evaluate the derivative it is useful to transform it to the reference element $\hat{\kappa}$. This way the length of all edges is unity, thereby making it unnecessary to consider explicitly different mesh sizes in different directions. Furthermore, the reference element is aligned along the coordinate axes, which makes it easier to decide on a refinement direction, once the characteristic direction of the derivative is known.
Using general mappings σ_κ the transformation of higher order derivatives is quite a complicated procedure. Therefore, general mappings will be decomposed into a linear part $\bar{\sigma}_\kappa$ and a higher order part $\tilde{\sigma}_\kappa$, such that $\kappa = \sigma_\kappa(\hat{\kappa}) = \tilde{\sigma}_\kappa\left(\bar{\sigma}_\kappa(\hat{\kappa})\right)$. For the transformation of derivatives only the linear part will be used, which leads to a simple transformation rule.
Introducing $\hat{\phi}$ as the representation of a real-space function ϕ on the reference element, such that $\hat{\phi}(\hat{\mathbf{x}}) = \phi(\mathbf{x}) = \phi\left(\sigma_\kappa(\hat{\mathbf{x}})\right)$, the derivative on the reference element can be obtained by applying the chain rule appropriately:

$$\frac{\partial \hat{\phi}}{\partial \hat{x}_j} = \frac{\partial \phi}{\partial x_i}\frac{\partial x_i}{\partial \hat{x}_j} = \frac{\partial \phi}{\partial x_i}\frac{\partial \left(\sigma_\kappa\right)_i}{\partial \hat{x}_j}, \tag{6.8}$$

where, according to the summation convention, it has to be summed over duplicated indices, in this case i.
Introducing the notation conventions $\hat{\partial}_k(\cdot) := \frac{\partial(\cdot)}{\partial \hat{x}_k}$ and $\partial_l(\cdot) := \frac{\partial(\cdot)}{\partial x_l}$ and recognising the term $\frac{\partial(\sigma_\kappa)_i}{\partial \hat{x}_j}$ as the Jacobian matrix J_{ij} of the transformation, (6.8) can be written in short as

$$\hat{\partial}_j \hat{\phi} = \partial_i \phi \, J_{ij}. \tag{6.9}$$

Denoting the nabla operator with respect to reference element coordinates $\hat{\mathbf{x}}$ with $\hat{\nabla}$ and the nabla operator with respect to real space coordinates with the usual ∇, the gradients on the reference element and in real space are connected through the Jacobian matrix of the transformation via

$$\hat{\nabla}\hat{\phi} = J^T \nabla \phi. \tag{6.10}$$

The transformation of higher order derivatives can be obtained by successive application of the chain rule. Already for the second order derivative, this will include derivatives of the Jacobian matrix. The second order derivative is given by

$$\hat{\partial}_k \hat{\partial}_l \hat{\phi} = \partial_i \partial_j \phi \, \hat{\partial}_k \left(\sigma_\kappa\right)_i \, \hat{\partial}_l \left(\sigma_\kappa\right)_j + \partial_i \phi \, \hat{\partial}_k \hat{\partial}_l \left(\sigma_\kappa\right)_i = \partial_i \partial_j \phi \, J_{ik} \, J_{jl} + \partial_i \phi \, \hat{\partial}_l J_{ik}. \tag{6.11}$$

Using the above mentioned decomposition of the mapping and assuming, that the non-linear part performs only small variations, but no change in the dimension of the element, i.e. $\tilde{\sigma}_\kappa(\bar{\mathbf{x}}) \approx \bar{\mathbf{x}}$, the terms containing derivatives of the Jacobian matrix, i.e. derivatives of the mapping of order 2 and higher, can be neglected, leading to a simple transformation equation for the second derivative:

$$\hat{\partial}_k \hat{\partial}_l \hat{\phi} = \partial_i \partial_j \phi \, J_{ik} \, J_{jl}. \tag{6.12}$$

Higher order derivatives can be obtained using the same scheme. Using only the linear part of the mapping, the transformation follows the same structure for all derivative orders: each index of the derivative tensor has to be contracted with the first index of the Jacobian matrix of the mapping. As a last example the transformation for the third

order derivative reads:

$$\hat{\partial}_l \hat{\partial}_m \hat{\partial}_n \hat{\phi} = \partial_i \partial_j \partial_k \phi \, J_{il} \, J_{jm} \, J_{kn}. \tag{6.13}$$

Anisotropic projection error estimates

Having transformed the approximated derivative at the center of an element κ to the reference element $\hat{\kappa}$ it remains to evaluate this tensor in order to decide, whether an anisotropic refinement should take place.
At this point it is necessary to come back to the basic projection error estimate. Equation (6.3) represents the general projection error, using a single value h to represent the mesh size. In an anisotropic environment it is necessary to find estimates, which employ the different mesh sizes of elements in order to improve the estimate and to make it useful for a decision on a refinement direction. Formaggia and Perotto [5] have developed anisotropic a priori error estimates and present anisotropic estimates of the interpolation error, however, their results are limited to linear polynomial approximations. Houston *et al.* [17] give a summary of the work done on generalizing these results for arbitrary order polynomial degrees. The most important result for the current context is given by the estimate

$$\|\phi - \Pi_p \phi\|_{L_2(\kappa)} \leq C \left[\int_\kappa \|\hat{D}^s(\mathbf{x})\|_F^2 d\mathbf{x} \right]^{1/2}, \quad 0 \leq s \leq \min(p+1, k) \tag{6.14}$$

where $\hat{D}^s$ is the sth order derivative tensor transformed to the reference element as decribed above, C is a constant which depends only on the dimension and the polynomial degree, k is the highest order of existing (weak) derivative of ϕ, i.e. $\phi \in H^k(\kappa)$, and $\|\cdot\|_F$ denotes the Frobenius norm, which for a tensor $A \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ is defined by

$$\|A\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} (A_{i_1 i_2 \ldots i_N})^2}. \tag{6.15}$$

If a tensor has a dominant component, its Frobenius norm can be efficiently reduced if only this component is reduced in size. The transformation of the derivative tensor from real space coordinates to the reference element reduces the derivative component(s) in the direction, which has a small mesh size. Consider, for example, a rectangular element aligned with the coordinate axes, having the lengths h_1 and h_2 along the x_1 and x_2 axis, respectively. For this case the Jacobian matrix reduces to the diagonal matrix

$$J = \begin{pmatrix} h_1 & 0 \\ 0 & h_2 \end{pmatrix}.$$

Considering the Hessian matrix $\mathcal{H}$ which represents the second order derivative tensor

$$\mathcal{H} = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix},$$

the transformed derivative reads

$$\hat{\mathcal{H}} = J^T \mathcal{H} \, J = \begin{pmatrix} h_1^2 H_{11} & h_1 h_2 H_{12} \\ h_1 h_2 H_{21} & h_2^2 H_{22} \end{pmatrix}.$$

Assuming, that the component H_{11} is the dominant component of the Hessian matrix, the Frobenius norm of $\mathcal{H}$ will be dominated by this component: $\|\mathcal{H}\|_F \gtrsim |H_{11}|$. The same holds for the transformed matrix, if the mesh sizes $h_1 \approx h_2$ are of similar magnitude. In order to reduce the projection error, the Frobenius norm of the transformed derivative has to be reduced, which can be done by reducing the mesh sizes. It is obvious that for the case of one dominant component it is most efficient to reduce the corresponding mesh size, in the stated example reducing only h_1 would have almost the same effect on reducing the error than reducing both h_1 and h_2.
Whereas the above reasoning was limited to rectangular, axis–parallel elements, the observations are true for general elements as well: If the derivative is exceptionally large in one direction, reducing the mesh size in that direction is an effective strategy, while reductions in the other directions are only of minor effect regarding the projection error. This offers a possibility of evaluating an element's aspect ratio. If the transformed derivative has components of similar magnitude in all directions, the element's aspect ratio is already optimal, further refinement should take place isotropically. If one component is dominant, however, the aspect ratio is not yet optimal and anisotropic refinement should be used in order to improve the aspect ratio. The evaluation process includes determination of possible dominant directions and comparison with threshold values before a decision on a possible anisotropic refinement can be made.

Evaluation for anisotropic indicators

For a second order derivative, i.e. the Hessian matrix, it is obvious to calculate the eigenvalues λ_i and the corresponding eigenvectors $\mathbf{v}_i$ of the tensor. Especially for the two-dimensional case this can be done explicitly with little effort. As a result, the ratio of the absolute values of the eigenvalues,

$$r = \frac{|\lambda_{i_m}|}{\max\limits_{\substack{j \neq i_m \\ 1 \leq j \leq d}} |\lambda_j|}, \qquad \text{with} \quad i_m \,:\, |\lambda_{i_m}| = \max_{1 \leq i \leq d} |\lambda_i|, \tag{6.16}$$

can be used as a measure of the magnitude of anisotropic properties of the solution. A value of unity describes isotropic behavior, whereas with increasing r the anisotropic characteristics get stronger. Anisotropic refinement should only be considered for an element, if the anisotropic ratio r is larger than an empirical threshold value θ_r.
The second important information gained from the calculation of eigenvectors is the direction, in which the derivative takes its maximum absolute value. If this direction is sufficiently aligned with one of the coordinate axes on the reference element, anisotropic refinement along this axis might be effective. In order to decide upon the refinement, the angle α_i between the eigenvector and the coordinate axis i has to be compared to a second threshold value θ_α. If the two conditions

$$r > \theta_r \qquad \text{and} \qquad |\alpha_i| < \theta_\alpha \tag{6.17}$$

are simultanously fulfilled the element should be flagged for anisotropic refinement along direction i.
For higher order derivatives the calculation of eigenvectors and eigenvalues is not possible. However, there still exist directions, in which the projected derivative takes on a maximum value. These could be found using an iterative scheme. The numerical evalu-

ation of the maximum could be quite expensive, however, so it seems reasonable to look for a simpler criterion. Concerning the refinement, there are very limited options, namely the refinement can be done along the coordinate axes only. Thus the exact knowledge of the characteristic direction is not necessary. It should suffice to calculate the ratio of the projected derivatives in the coordinate directions on the reference element. As these are found to be the terms on the main diagonal, this is a trivial evaluation. This leads to a simple criterion, which is based on only one empirical threshold factor $\tilde{\theta}_r$. If there is an i, $1 \leq i \leq d$, such that the condition

$$|\hat{\partial}_i^{p+1}\hat{\phi}| > \tilde{\theta}_r \max_{\substack{j \neq i \\ 1 \leq j \leq d}} |\partial_j^{p+1}\phi| \tag{6.18}$$

is satisfied, the refinement should take place anisotropically along the $\hat{x}_i$-axis[10].
Apart from this very simple criterion there are, of course, other possibilities to evaluate higher order derivatives. As mentioned above, a numerical algorithm to search for the direction with maximal derivative value could be used. To make such an algorithm more efficient, it could be limited to a subset of directions with only small deviation from the coordinate axes, this can be compared to the second condition in (6.17).
A further approach is based on interpreting the third order derivative as d second order derivatives of the individual components of the gradient. "Blending" these Hessian matrices would result in a single matrix that can be evaluated using the above mentioned algorithm for second order derivatives. A description of how Hessian matrices can be blended is given in Chapter B in the appendix. Alternatively the second order derivatives could be evaluated separately. Subsequently the element would be flagged for anisotropic refinement if all or "most" of the individually evaluated indicators coincide.
Such ideas can be extended to even higher order derivatives. The more natural way, however, seems to be using only the term on the main diagonal of the tensor, which is a very cheap operation. It has to be kept in mind that anisotropic refinement will only be a useful addition to adaptive algorithms, if the evaluation of anisotropic indicators is inexpensive.

Application to vector valued solution functions

So far only scalar valued solution functions ϕ have been considered. Fluid flow calculations include vector valued solution functions $\mathbf{u}$ which are composed, for example, of the density, the momentum components and the internal energy. Based on the described algorithm, three different treatments of multi-component functions are possible.

a) The refinement indicator is evaluated separately for each component. If a sufficient number of the individual indicators suggest the same refinement strategy, and the other indicators do not contradict this suggesetion (i.e. the other indicators suggest isotropic refinement), this strategy is chosen, otherwise the element is refined isotropically.

b) The information coming from the derivatives of the individual vector components is "blended" before it is evaluated. In case of second order derivatives a procedure

[10] A similar criterion is used by Aftosmis and Kroll [1], who use an indicator based on undivided differences, which correspond to the transformed derivatives in the main directions of the reference element. Using the ratio of the values for individual coordinate directions, they achieve promising results.

called *metric intersection* can be applied to the Hessian matrices, which would have to be appropriately scaled in order to represent relative changes instead of absolute ones, see Chapter B in the appendix. A simpler alternative would be to add in a vectorial sense the characteristic directions resulting from the individual components. In order to use information on the magnitude of anisotropic effects, the characteristic directions, i.e. eigenvectors in case of second order derivatives, could be scaled by the anisotropic ratio r defined in Equation (6.16).

c) Instead of several independent components, one key variable is used to determine the properties. It is possible to choose one of the conservative variables ρ, ρv_i or ρe. However, in order to deliver good estimates of the anisotropic features, the key variable should be characteristic of the whole flow field. Therefore derived variables like the total velocity scalar v or the Mach number M could be better choices. This technique overcomes the need to decide upon an empirical combination of several indicators or derivative tensors. It is, therefore, easier and in particular cheaper to evaluate.

Of course it is also possible to use a combination of the presented approaches. It might be useful, for example, to use not only one, but several key variables to decide upon the anisotropic features. In that case, the ideas presented under a) and b) will have to be applied to the chosen key variables instead of the conservation variables.

Review of assumptions

The most restrictive assumption in using the described derivative indicators is the smoothness of the solution ϕ. For the application of the $(p+1)$th order derivative it has to be assumed, that $\phi \in H^k$, with $k \geq p+1$. While this is a reasonable assumption in boundary layers, it will generally not be fulfilled in other regions of the flow, especially in interior layers like shocks. It has to be noted, however, that the described anisotropic refinement will be most effective in boundary layers, anyway. This is caused by the alignment of the mesh with the characteristic directions of the boundary layer. Concerning shocks, the mesh will generally not be well aligned with these directions, resulting in an isotropically dominated refinement. Therefore, the described indicators should yield good results for the Navier–Stokes equations. For problems dominated by shocks, especially for the Euler equations which do not result in strong boundary layers, it might be considered, however, to use the first order derivative, independent of the polynomial degree.
Concerning a possible extension to *hp*–refinement it has to be assumed that the polynomial degree p will be small in the vicinity of shocks. Thus the applicability of derivative indicators will not be as problematic as it might seem, even near interior layers.

6.2.3. Other sources of information on anisotropic features

Apart from the described indicators, other sources of information on anisotropic features are used by several authors. Concerning triangular, linear Finite Elements considerable work has been done using the information of approximated second order derivatives and the resulting mesh metric field directly in remeshing algorithms, see for example the work by Formaggia *et al.* [4], Frey and Alauzet [6, 7], Huang [18] or Sahni *et al.* [19]. This approach has the advantage, that no additional step is needed for the evaluation of anisotropic features after the error estimation, as the metric field combines both informations. However, the error estimation is limited to the feature based approach of

considering only the interpolation error. In order to incorporate more appropriate error estimation techniques, Venditti and Darmofal [21] have combined the metric approach with adjoint based error indicators, resulting in an approach of using weighted metrics. Still, this technique requires a complete remeshing and is therefore not applicable in the framework of the application considered in this work, which is based on a hierarchic structure of triangulations.

Sun and Wheeler [20] suggest several global refinements, each using anisotropic refinement in a fixed direction for all elements. Obtaining the solution on all these discretizations, error indicators can be calculated for the refined elements. Choosing, individually for each element, the refinement strategy which results in a higher reduction of the estimated error yields quite optimal refined meshes. However, the effort of solving the problem several times (normally d times) on globally refined meshes is considerable. Sun and Wheeler use the obtained meshes for several time steps in the calculation of transient problems. In that case, the effort might be affordable, whereas ist does not seem to be justified for the numerical approximation of stationary flow problems.

Trying to reduce the computational effort of this approach, Houston *et al.* [17] suggest to solve local problems. Only the element under consideration is refined according to the different anisotropic refinement options. Afterwards, for each of these alternatives an improved numerial solution is obtained only locally on this element, assuming, that the solution on the surrounding elements does not change much. Comparing the reduction in the estimated error of the different alternatives the best strategy can be chosen. This technique offers the great advantage that it does not depend on any a priori information on the error. Furthermore, it is possible to use goal oriented error indicators, not only for flagging the elements to be refined, but also in the evaluation of an anisotropic refinement direction. These advantages might balance the fact, that the evaluation of this indicator – including the solution of local problems on each element considered for refinement – is still quite expensive.

7. Numerical results

It can be expected that the anisotropic indicators described in Section 6.2 are able to reduce the number of elements needed to achieve a certain accuracy in the target functional values. The practical performance is unknown, however, as is the effectiveness of the different concepts. Furthermore, the best choice of the empirical threshold factors introduced into the indicator evaluation is unknown as well. Therefore, this chapter will provide some numerical examples in order to evaluate the effectiveness of the individual indicators.

As the mesh is aligned with the boundary, it can be expected that anisotropic refinement is quite effective in boundary layers, but perhaps less so in interior layers (shocks), as those are, in general, not aligned with the elements, leading to characteristic anisotropic directions which do not coincide with the elements' orientation. As the node positions are not changed by an h–refinement, interior layers will probably exhibit a high fraction of isotropically refined elements.

After a brief description of the different test cases in Section 7.2, the individual results will be given in Sections 7.3 and 7.4 for the jump and derivative indicators, respectively. Finally, a comparison between the indicators and recommendations based on the numerical examples are given in Section 7.5. Prior to these aspects, some general remarks on the expected convergence behavior will be given in the following section.

7.1. Expected convergence behavior

Using a numerical method with fixed degree of the local piecewise polynomial approximation, which is the case for pure h–refinement, the asymptotic rate of convergence is determinded by the method itself, not by a refinement strategy. This means, that neither adaptive refinement nor anisotropic refinement can improve this rate. However, starting with quite coarse meshes, the solution does not behave according to this asymptotic rate in the first refinement steps. For many problems arising from engineering tasks, the required accuracy is gained before the asymptotic range is reached. Using elaborate refinement techniques it might be possible to improve the constant involved in the expression of the asymptotic error estimate as well as to reach the asymptotic region earlier.

The anisotropic indicators developed in this work are intended to be used in combination with an adaptive refinement strategy. Thus, the only reasonable reference values for evaluating the efficiency of anisotropic refinement are the values obtained with an according isotropic adaptive refinement.

Starting from the same initial mesh and solution on this mesh, the anisotropic indicator evaluation leads to a reduced number of elements in the next refinement step, as some refinement flags are changed from isotropic to anisotropic refinement. However, no additional elements are flagged for refinement if the same refinement and coarsening fractions f_r and f_c as introduced in Section 4.2 are used. This behavior is intended, as it reduces the number of elements on the next level, hopefully without influencing the accuracy too

much. This technique yields a discrete function space, that is embedded in the isotropic one. This means, that any function on the anisotropic mesh can also be represented on the isotropic one, but not vice versa. Assuming, that a richer function space yields a better approximation, at least not a worse one (the *saturation assumption*), the accuracy of the approximated solution on the anisotropically refined mesh will generally be worse than that on the isotropically refined reference mesh. If the indicators are selected according to physical features, however, the difference will be small, leading to an only slightly worse approximation on a mesh with considerably less elements.
Following the adaptive process for more than one refinement step, the two methods do not start from the same mesh, but from the individually obtained refined meshes for the second refinement step. In case of the anisotropically refined mesh, a smaller fraction of the elements will be located in layers as compared to the isotropically refined mesh. It is quite likely that elements in the layers have large errors in the second refinement step. If these elements form a smaller fraction, additional elements outside the layer may also be flagged, leading to a better overall approximation. In summary, as the adaptive refinement process does *not* start from the same mesh in each step for isotropic and anisotropic refinement, the anisotropic refinement may, in the course of several adaptive refinement steps, eventually lead to approximations with higher accuracy as compared to the isotropically refined version using the same number of refinement steps.

7.2. Test cases

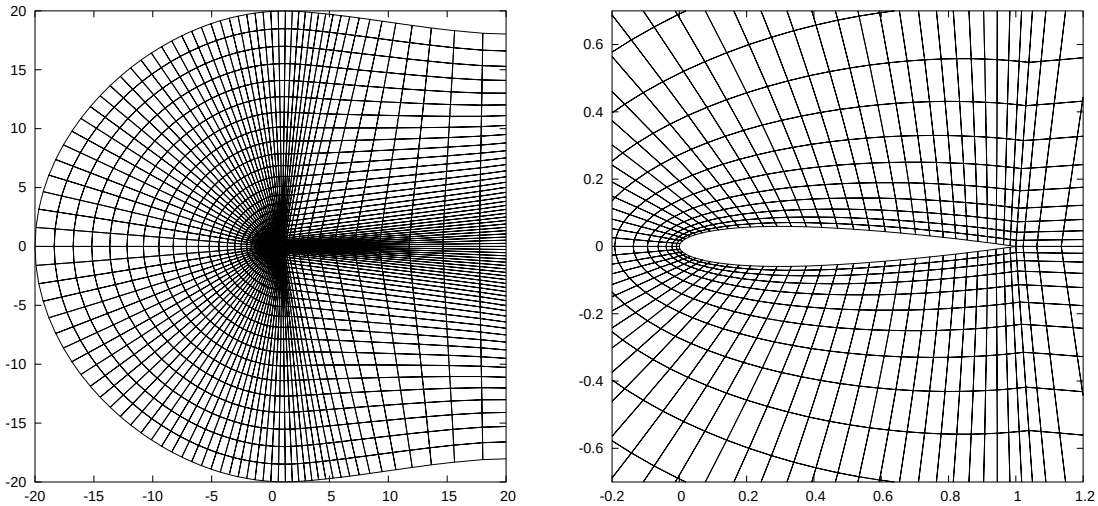
In order to test the anisotropic refinement process and the developed indicators for different typical aerodynamic flow situations, several test cases are used, ranging from sub- to supersonic flow fields. As the code used to perform the calculations (see Chapter A in the appendix) is currently only capable of laminar flow calculations, only low Reynolds number flows can be computed based on the compressible Navier–Stokes equations, which include viscous terms. The transonic test case is computed based on the compressible Euler equations which do not exhibit viscous effects.
The following test cases are evaluated using the DG(1) method which is second order accurate. In order to evaluate the applicability of the developed indicators to higher order discretizations as well as the influence of the order of the numerical scheme on the performance of anisotropic mesh refinement some computations have been performed also for the DG(2) method which is third order accurate.
The discrete equations are solved using a Newton algorithm for the nonlinear problem with a restarted version of the GMRES algorithm for the linear subproblems, see [9, 14, 16]. GMRES is also used for the linear dual problem. The trans- and supersonic test cases use a shock capturing based on an artificial viscosity term, which is consistent as it vanishes for the exact solution, for details see [9, 10].

7.2.1. NACA0012

All test cases presented in the following sections are based on the NACA0012 airfoil, which is given by the equation

$$y(x) = \pm 0.6 \left(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4\right). \tag{7.1}$$

As this equation yields an airfoil with slightly more than unit length, a rescaling is used to ensure that $y(0) = y(1) = 0$.
The starting mesh for the adaptive process is a C–type mesh containing 3072 elements. This mesh as well as a zoom of the mesh in the vicinity of the airfoil is shown in Figure 7.1.



(a) complete domain

(b) vicinity of the airfoil

Figure 7.1.: Initial mesh around a NACA0012 airfoil of unit length, 3072 elements.

For some test cases a coarser version of this mesh is used, which is obtained by a global coarsening procedure combining four elements of the fine mesh to one on the coarse mesh. This results in a total number of 768 elements.

7.2.2. Test case A: Viscous subsonic flow

The subsonic viscous laminar flow over a NACA0012 airfoil at Mach number $M = 0.5$, Reynolds number $Re = 5000$ and zero angle of attack is computed based on the compressible Navier–Stokes equations. As the airfoil is symmetric, the lift coefficients vanish for zero angle of attack. Figure 7.2 shows the Mach isolines and the streamlines of the resulting symmetric flow.
The target functional under consideration is the pressure induced drag coefficient c_{dp}, which is evaluated to approximately 0.0222875 by fine grid computations.

7.2.3. Test case B: Inviscid transonic flow

The transonic inviscid flow over a NACA0012 airfoil at Mach number $M = 0.8$ and an angle of attack of $\alpha = 1.25°$ is computed based on the compressible Euler equations. Figure 7.3 shows the Mach isolines and the streamlines of this flow. The subsonic flow is accelerated to supersonic velocities near the surface of the airfoil. On the second half of the upper airfoil surface the velocity is reduced spontaneously by a shock, influencing both drag and lift. This shock on the upper side of the airfoil as well as the smaller one at the lower side can clearly be seen by the agglomeration of Mach isolines.

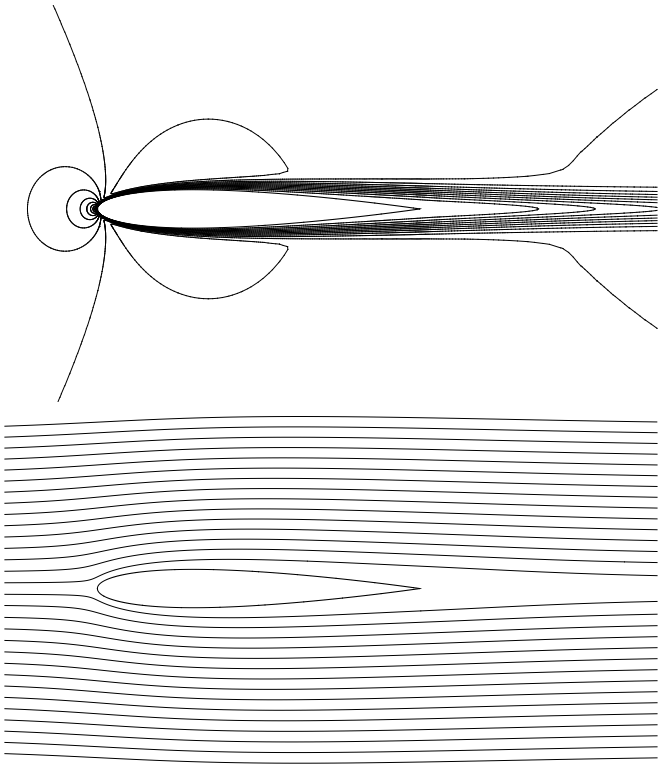


Figure 7.2.: Test case A: viscous flow at $M = 0.5$, $\alpha = 0°$ and $Re = 5000$: Mach isolines (top) and streamlines (bottom).

The target functional under consideration is the pressure induced drag coefficient c_{dp}, which is evaluated to approximately 0.0219 by fine grid computations.

7.2.4. Test case C: Viscous supersonic flow

The supersonic viscous laminar flow over a NACA0012 airfoil at Mach number $M = 1.2$, Reynolds number $Re = 1000$ and zero angle of attack is computed based on the compressible Navier–Stokes equations. A detatched bow shock in front of the airfoil decelerates the flow to subsonic velocity in the vicinity of the airfoil. However, the main flow accelerates to supersonic velocities again, whereas the flow on the surface of the airfoil and in the wake stays subsonic. Figure 7.4 shows the Mach isolines and the streamlines for this test case. From the streamlines its hard to recognize the supersonic character of the flow, but in front of the bow shock the streamlines are absolutely parallel, as is typical of a supersonic freestream, in which no information can be passed upstream and thus no deformation of streamlines in front of obstacles can occur.
The target functional under consideration is the viscous drag coefficient c_{df}, which is evaluated to approximately 0.1079 by fine grid computations.

7.3. Results for the jump indicator

The evaluation of the jump indicator includes one empirical threshold value θ, see (6.2), which describes the desired ratio of the largest average jump in one direction and the average jumps in the remaining direction(s). A small value of this threshold value will

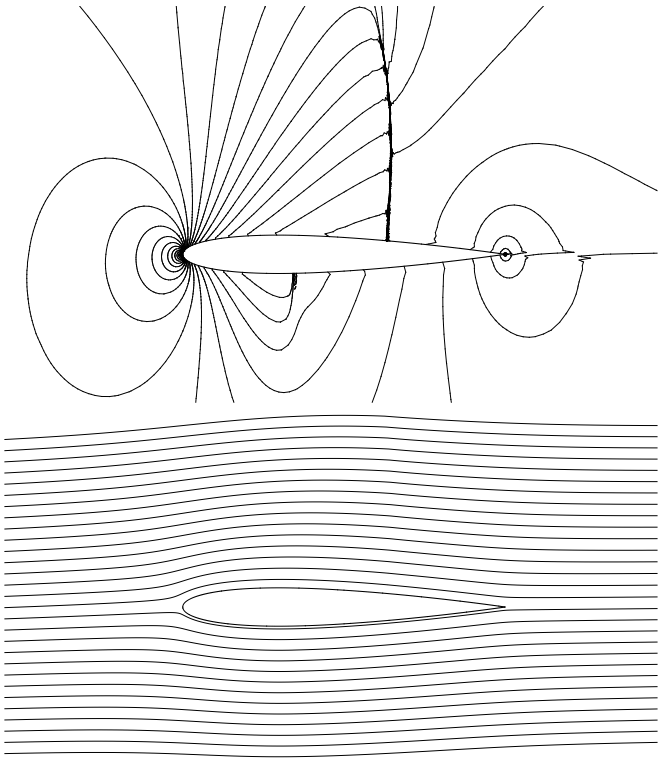


Figure 7.3.: Test case B: inviscid flow at $M = 0.8$ and $\alpha = 1.25°$: Mach isolines (top) and streamlines (bottom).

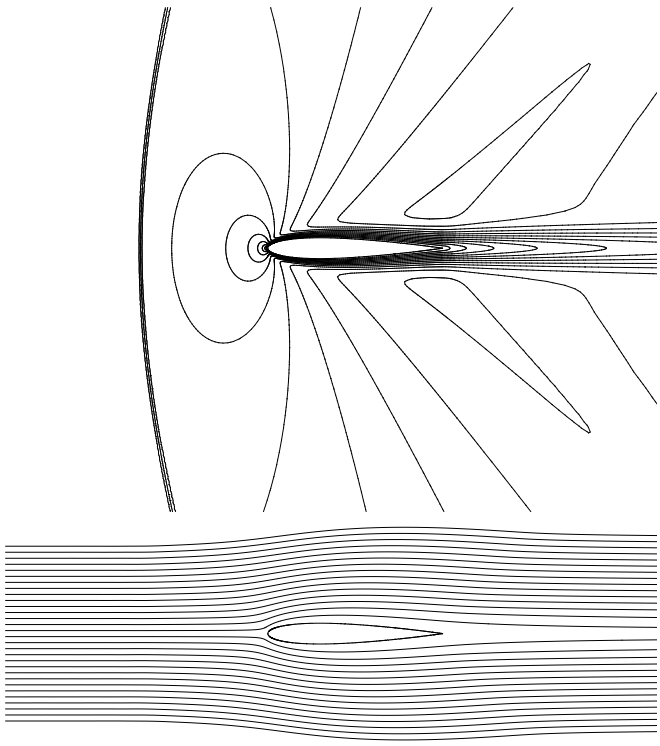


Figure 7.4.: Test case C: viscous flow at $M = 1.2$, $\alpha = 0°$ and $Re = 1000$: Mach isolines (top), and streamlines (bottom).

result in a lot of elements flagged for anisotropic refinement and thus strongly reduce the number of elements on the next refinement level. However, if the value is too small, several elements will be refined anisotropically although isotropic refinement might be more appropriate. This leads to a reduced approximation accuracy. In order to avoid this effect, θ must be chosen sufficiently high. On the other hand, too large values will prevent anisotropic refinement on elements for which this strategy might be appropriate, thus increasing the number of elements required for obtaining a certain accuracy. The optimal choice of the threshold value varies slightly for different test cases and initial meshes. However, the following observations were similar for other test cases. Figure 7.5 shows the convergence of the error in the target functional values for test case A, using adjoint based error indicators. The individual curves are obtained using values of θ ranging from 2.0 to 4.0. For comparison, also the values for a pure isotropic refinement are given.

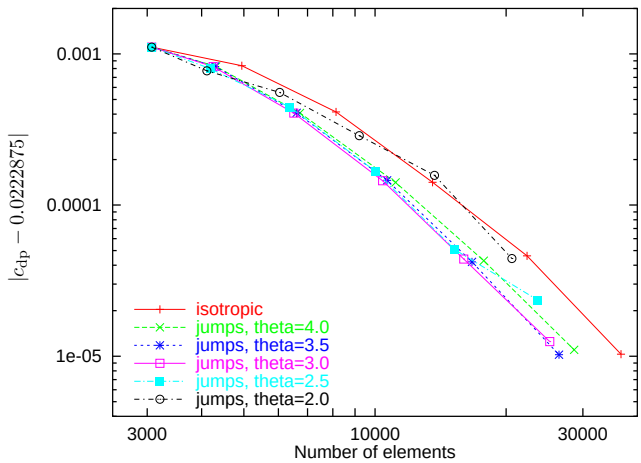


Figure 7.5.: Test case A: Convergence of the error in c_{dp} for the DG(1) method using adjoint based error indicators combined with anisotropic jump indicators.

Clearly, the choise $\theta = 2.0$ is too small, as it leads to no significant reduction in the number of elements needed for a certain accuracy and even crosses the curve given by isotropic refinement in one point. Apart from that, the differences for various values of θ are small, in the given case a value of $\theta = 3.0$ or $\theta = 3.5$ seems optimal. The accuracy achieved after a given number of refinement steps is always very similar to that of isotropic refinement after the same number of adaptive steps, thus the aim of reducing the number of elements while obtaining similar accuracy is reached. Using the jump indicator the number of elements on the finest level considered in Figure 7.5 can be reduced by approximately 30 %.

The applicability of the jump indicator to higher order DG methods is evaluated using the DG(2) scheme for test case A. As the number of degrees of freedom per element is higher by a factor of $\frac{9}{4}$ as compared to the DG(1) method, the coarser initial mesh is used for this subcase. Figure 7.6 compares the target functional error obtained with a value of $\theta = 3.0$ with the respective errors of an isotropic refinement.

The graph shows both the improved convergence of higher order methods and the applicability of the jump indicator to these higher order approximations. The comparison between isotropic and anisotropic refinement is very similar to that for the DG(1) method, in this case more than 40 % of the elements can be saved on the finest mesh. It is worth

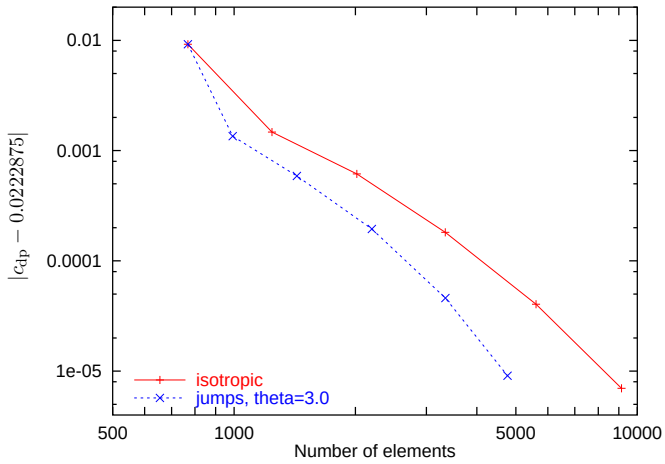


Figure 7.6.: Test case A: Convergence of the error in c_{dp} for the DG(2) method using adjoint based error indicators combined with an anisotropic jump indicator.

mentioning one additional aspect: As the initial mesh is quite coarse, the convergence in the beginning does not fit in the general curve it follows afterwards. From the initial mesh to the first refined one there is a significant drop in the target functional error, indicating that the improvement of the mesh is considerable. After this initial refinement step the following ones are well within the expected convergence behavior, indicating that the asymptotic range is reached already, all individual points for actual calculations lie on a smooth curve.

Figure 7.7 shows a comparison of the efficiency of the residual based error indicator compared to the adjoint based indicator, again for the DG(1) method. Clearly, the convergence of the target functional is improved significantly if adjoint based error indicators are used to select the elements for refinement. This fact justifies the additional cost associated with the solution of the adjoint problem.

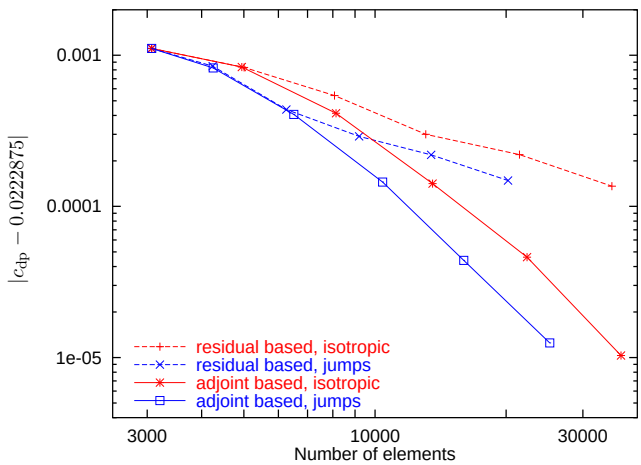


Figure 7.7.: Test case A: Convergence of the error in c_{dp} for the DG(1) method using residual and adjoint based error indicators combined with an anisotropic jump indicator.

For both error indicators the anisotropic refinement based on jumps of the solution effectively reduces the number of elements needed for a given accuracy as compared

to the respective isotropic refinement case. Using residual based error indicators the achieved saving of elements amounts to about 40 % and is thus slightly higher than in the adjoint based case.

7.4. Results for derivative indicators

The evaluation of the derivative indicators described in Section 6.2.2 offers a wide range of alternatives. In order to show only the relevant results, no comparison of different threshold values will be shown. The main results will be shown for test case A using the DG(1) method and the adjoint based error indicators.

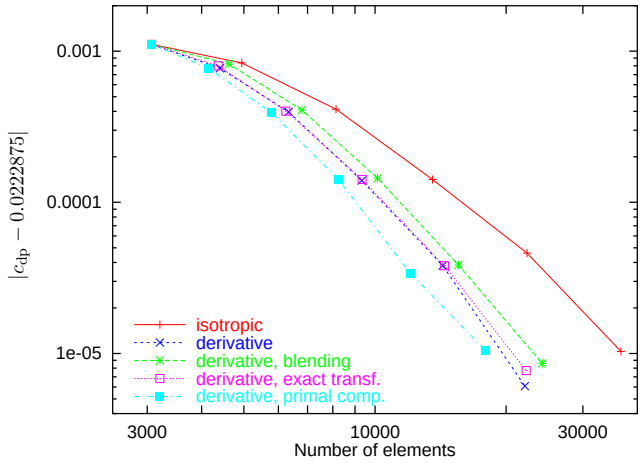


Figure 7.8.: Test case A: Convergence of the error in c_{dp} for the DG(1) method using adjoint based error indicators combined with anisotropic derivative indicators.

Figure 7.8 shows several examples for the target functional convergence using different derivative indicators. The reference case is based on evaluating the characteristic direction of the derivative (in this case the eigenvector belonging to the larger absolute eigenvalue of the Hessian matrix) and comparing it to a threshold angle $\theta_\alpha = 15°$, see (6.17), as well as comparing the anisotropic ratio as defined in (6.16) to a second threshold value $\theta_r = 3.0$. This is done individually for all conservative variables. If at least two of these evaluations suggest the same anisotropic refinement and the others do not suggest a different direction, i.e. they suggest isotropic refinement, an element is flagged for anisotropic refinement in this particular direction. The resulting convergence curve clearly shows, that the accuracy reached after a given number of refinement steps is very similar to that of the corresponding isotropic refinement, but uses a reduced number of elements, in the example this reduction rises to about 40 % after the last refinement step. As described in Section 6.2.2, the derivative indicator uses a linearized transformation from real space coordinates to the reference element. The assumption, that higher order terms including derivatives of the Jacobian matrix of the transformation can be neglected seems justified. Using the exact transformation given by (6.11), the overall convergence changes only slightly, as shown in Figure 7.8. This justifies the use of the linearized transformation, which enables a simple and unified approach for all derivative orders.
The concept of blending is introduced in Section 6.2.2 as a possible means of dealing with vector valued solution functions. The general idea is to use a combination of the individual derivative components which is dominated by the strongest anisotropic features

and thus allows to refine an element, if one variable shows strong anisotropic features whereas the others do not. However, as Figure 7.8 indicates, the expected additional gain concerning the number of elements cannot be observed in the numerical examples, in contrast the blended derivatives lead to a slightly worse convergence behavior. This effect might be due to the fact, that the approximate metric intersection described in the appendix is too inaccurate. The approximation tends to give results with a too small anisotropic ratio. However, simply lowering the corresponding threshold value does not solve the problem. Therefore, until higher quality approximations of metric intersections are available, the use of blending does not seem to be useful and the additional effort for the intersection procedure can be saved.
The last indicator used in Figure 7.8 is based on evaluating the derivative only in the direction of the coordinate axes on the reference element (primal directions) and comparing the resulting ratio to a threshold value $\tilde{\theta}_r$, see (6.18). No threshold angle θ_α is used. Using both threshold values θ_r and θ_α from above, a corresponding threshold value can be calculated for the primal directions, if the derivative described by these threshold values is projected to the coordinate axes of the reference element. For a second order derivative this results in

$$\tilde{\theta}_r = \frac{\theta_r \cos^2(\theta_\alpha) + \sin^2(\theta_\alpha)}{\cos^2(\theta_\alpha) + \theta_r \sin^2(\theta_\alpha)}. \tag{7.2}$$

However, this value is not unique to the combination of threshold ratio and threshold angle. Using $\theta_r = 3.0$ and $\theta_\alpha = 15°$ as above, $\tilde{\theta}_r = 2.53$ is obtained. This value can also be produced by a lower threshold ratio for lower threshold angles and by a higher threshold ratio for higher threshold angles, as can be seen in Figure 7.9.

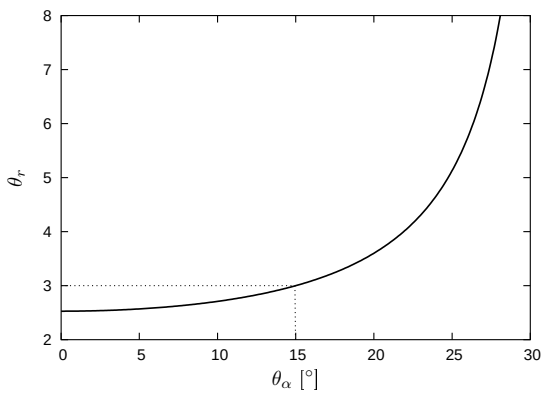


Figure 7.9.: Combinations of threshold ratio θ_r and threshold angle θ_α resulting in the modified threshold ratio $\tilde{\theta}_r = 2.53$.

This effect leads to a number of elements being flagged for anisotropic refinement, which were not flagged using the more involved indicator based on both threshold ratio and angle. Obviously, this results in a slightly improved convergence behavior. However, on the finest mesh the achieved accuracy is slightly worse than for all other methods. Still, the differences are small and thus justify the use of this simplified criterion, which is directly applicable to higher order derivatives as well.
Using different threshold values the results vary, as can be expected. Increasing the threshold angle θ_α from 15° to 20° results in a steeper descent of the convergence curve after the first refinement steps. However, in the later refinement steps the error changes the sign, afterwards the absolute value of the error starts growing again. This behavior

indicates, that the refinement is not really appropriate for the problem, i.e. that the threshold values are not strict enough. Similar results are obtained using a key variable like the Mach number M or the velocity scalar v. In early refinement steps the error is reduced more quickly than with the indicators using all conservative variables, but later on the convergence behavior gets worse, the error changes its sign and grows in absolute values. Again, this indicates, that the chosen indicator does not catch all aspects of the physical behavior of the flow under consideration, i.e. the flow field and its anisotropic features cannot be represented with sufficient accuracy by a single key variable.
Similar as in the case of the jump indicator, two further comparisons are presented using test case A. Figure 7.10 shows the application of derivative indicators to a higher order method.

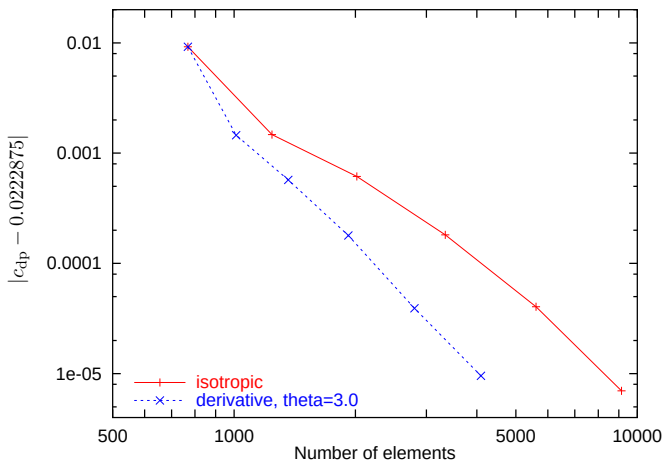


Figure 7.10.: Test case A: Convergence of the error in c_{dp} for the DG(2) method using adjoint based error indicators combined with an anisotropic derivative indicator.

For the DG(2) method the tensor of third order derivatives is evaluated instead of the Hessian matrix, which is used for the deriavtive indicator applied to the DG(1) method. As described above, the application to higher order methods implies the use of a linearized transformation and evaluating only the projected derivative in the primal directions of the reference element. The convergence curve shown in Figure 7.10 is obtained using $\tilde{\theta}_r = 3.0$, but other values show only minor variations of the behavior. On the final mesh, about 50 % of the elements can be saved using a derivative indicator for anisotropic refinement.
Finally, a comparison between residual based and adjoint based error indicators is shown in Figure 7.11, using the DG(1) method, an anisotropic derivative indicator with $\theta_r = 3.0$, $\theta_\alpha = 15°$ and a linearized transformation.
The resulting behavior can be clearly seen. In both cases anisotropic refinement based on the derivative indicator saves a significant number of elements while obtaining similar accuracy as compared to the respective isotropic refinement. In the case of residual based error indicators the accuracy is even improved noticably. The last refinement step for the isotropic case results in an error comparable to the error of the last but one anisotropic refinement step. The saving of elements amounts to over 60 % in this case and is thus higher than in the case of adjoint based error indicators. However, employing adjoint based error indicators yields a significantly improved convergence of the target functional values and thus proves to be advantageous.

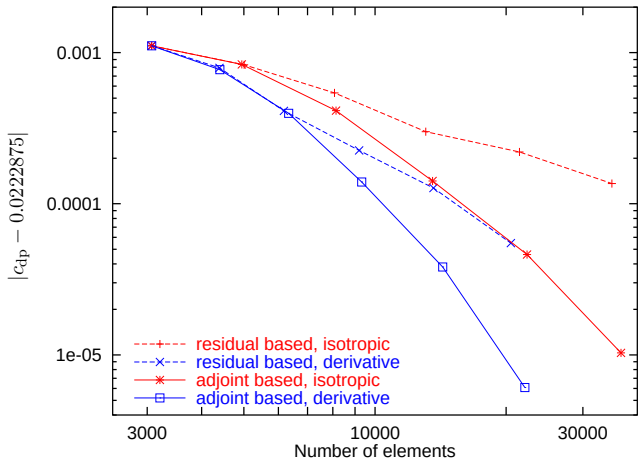


Figure 7.11.: Test case A: Convergence of the error in c_{dp} for the DG(1) method using residual and adjoint based error indicators combined with an anisotropic derivative indicator.

7.5. Comparison and indicator recommendation

In order to compare the efficiency of jump indicators and derivative indicators, some of the above examples for test case A will be shown in a common plot. Additionally, some results will be shown for the other test cases as well.

7.5.1. Comparison for test case A

As the above examples show that the use of adjoint based error indicators is clearly advantageous, only the corresponding results will be repeated here. Concerning results obtained with residual based error indicators, Figure 7.7 can be compared to Figure 7.11. Figure 7.12 shows the convergence behavior for test case A, using the DG(1) method, where a jump indicator with $\theta = 3.0$ and a derivative indicator based on the conservative variables with $\theta_r = 3.0$ and $\theta_\alpha = 15°$ is used.

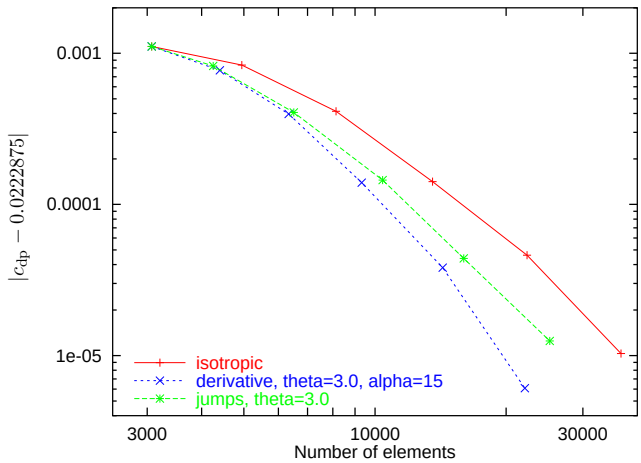


Figure 7.12.: Test case A: Convergence of the error in c_{dp} for the DG(1) method using adjoint based error indicators combined with anisotropic jump and derivative indicators.

Whereas the accuracy reached after a given number of refinement steps is very similar for both versions of anisotropic refinement, the number of elements is smaller for the derivative indicator, i.e. the jump indicator reduces the use of resources less effectively. However, both versions demonstrate an improved behavior as compared to isotropic refinement. As the time needed for the evaluation of the indicators can be neglected in comparison with the time spent on the solution process, the saving of elements directly translates to a gain in computational time. Furthermore, a smaller number of elements also reduces the memory consumption.
Figure 7.13 shows the corresponding comparison for the DG(2) method. Here, the jump indicator is evaluated using $\theta = 3.0$, whereas $\tilde{\theta}_r = 3.0$ is used for the derivative indicator.

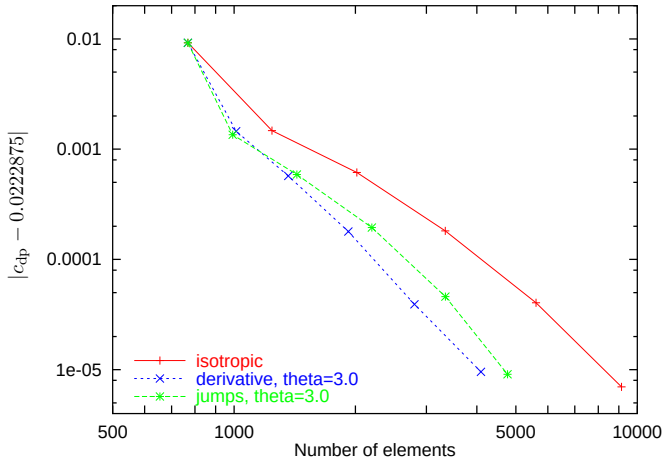
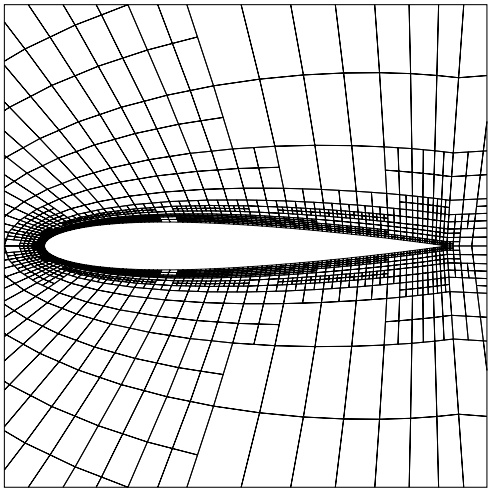


Figure 7.13.: Test case A: Convergence of the error in c_{dp} for the DG(2) method using adjoint based error indicators combined with anisotropic jump and derivative indicators.

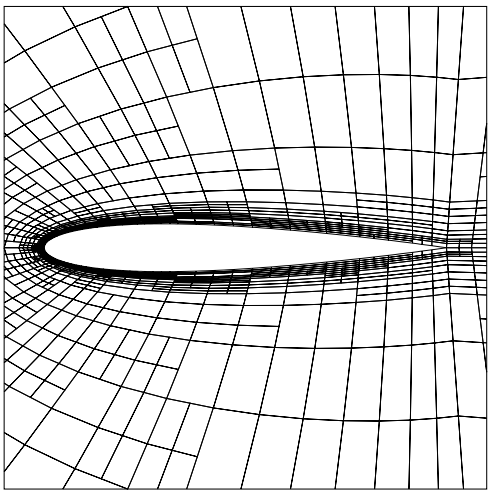
Both indicators perform similarly well, the convergence of the derivative indicator is again slightly better than that of the jump indicator. Compared to the DG(1) method, a similar or even larger fraction of elements can be saved on the finest mesh.
Examples of the adapted meshes are shown in Figure 7.14. It shows meshes in the vicinity of the airfoil after three adjoint based refinement steps for the DG(2) method. The comparison demonstrates the good resolution of the boundary layer in all cases. Near the leading edge, isotropic refinement is used even if anisotropic refinement is enabled. Due to the strong curvature of the airfoil, the flow changes rapidly along the airfoil as well as in the orthogonal direction. However, further downstream the curvature reduces and anisotropic refinement is appropriate. In this region isotropic refinement produces an unnecessary amount of elements as the element size orthogonal to the airfoil surface is reduced. The comparison between the adapted meshes based on the jump and the derivative indicator shows that the resulting refinement strategy is quite similar in both cases.

7.5.2. Comparison for test case B

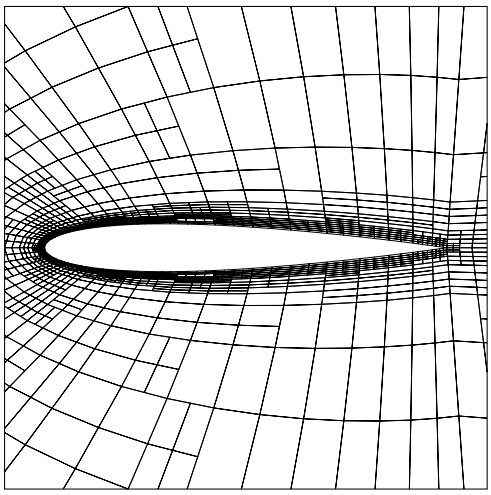
As inviscid flows do not exhibit strong boundary layer effects, the occurance of anisotropic effects is mainly limited to shocks. The solution for the transonic test case B has a large shock on the upper side of the airfoil and a smaller one at the lower side. In the region of this shock anisotropic refinement is advantageous. The shock represents a discontinuity



(a) isotropic refinement



(b) anisotropic refinement based on derivative indicators



(c) anisotropic refinement based on jump indicators

Figure 7.14.: Test case A: Adapted mesh after three refinement steps using adjoint based error indicators for the DG(2) method.

however, thus the assumption that the solution is sufficiently smooth no longer holds in this region. As the derivative indicator is based on this assumption, its performance is likely to deteriorate. In fact, the derivative indicator based on the conservative variables used for test case A does not lead to any improvement of the convergence behavior. However, using the Mach number M as key variable, anisotropic refinement governed by a derivative indicator can be used effectively. Whereas the jump indicator does not suffer from unsatisfied assumptions, anisotropic refinement is still restricted to those parts of the domain, where the anisotropic features are sufficiently aligned with the mesh.
Figure 7.15 shows a comparison of the convergence behavior for this derivative indicator based on the Mach number and isotropic refinement, as well as the jump indicator. The DG(1) method was used in all cases.

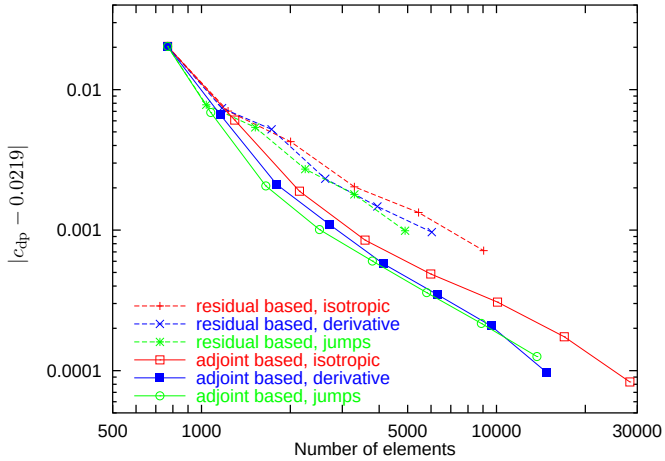
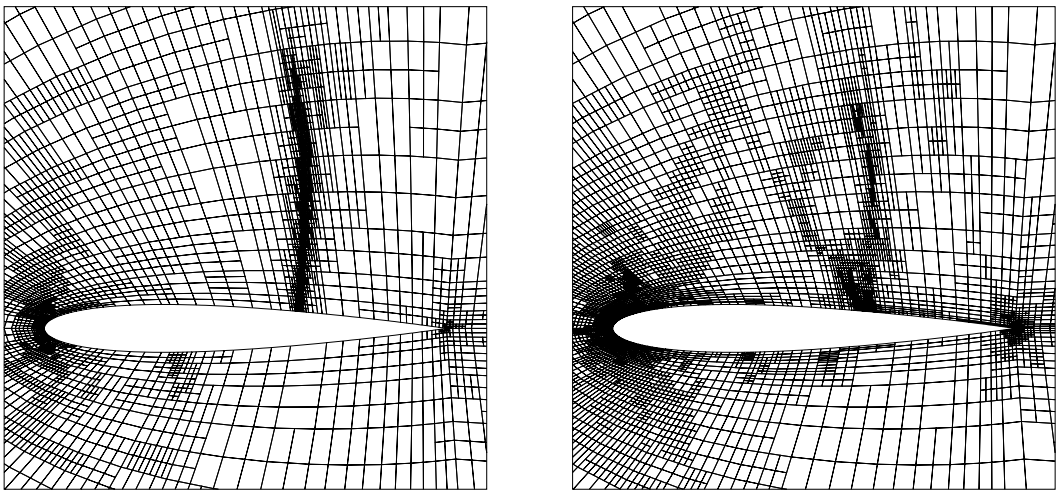


Figure 7.15.: Test case B: Convergence of the error in c_{dp} for the DG(1) method using residual and adjoint based error indicators combined with anisotropic jump and derivative indicators.

The computations have been started on the coarser initial mesh, thus more refinement steps are used compared to test case A. Given the drawbacks of the applicability to inviscid and transonic flows, both indicators perform quite well and lead to a reduction of about 40 % in the number of elements on the finest mesh in case of adjoint based error indicators. Although this is a surprisingly high number, it has to be kept in mind, that the number of refinement steps is higher for this test case. The accuracy reached after a given refinement step is very similar for all presented methods and the jump indicator performs slightly better. The only exception to these two observations is the finest mesh, on which the jump indicator leads to a reduced accuracy.
Concerning residual based error indicators the convergence of the target functional error is slightly worse, as might be expected from the results of test case A. For test case B, anisotropic refinement is not as effective for residual based indicators as for adjoint based indicators, although an improvement can still be achieved. Again, the jump indicator performs slightly better than the derivative indicator.
Examples of adapted meshes are given in Figure 7.16, which shows the anisotropic meshes obtained after six adaptive refinement steps using the derivative indicator.
The comparison between the meshes obtained with residual based and adjoint based error indicators is impressive. Whereas the first strategy refines mainly the shock on the upper airfoil surface and the leading edge, the latter reduces the refinement of the shock in favor



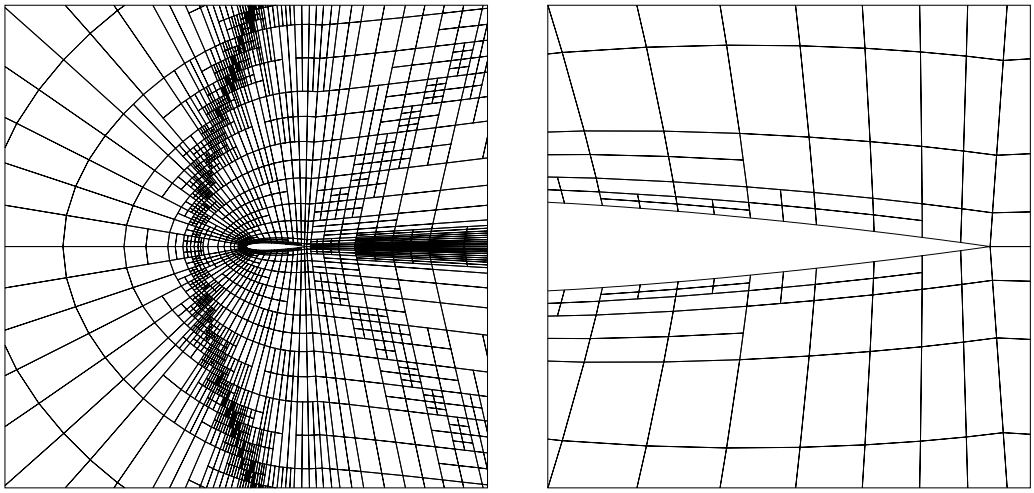
(a) residual based error indicators (b) adjoint based error indicators

Figure 7.16.: Test case B: Adapted anisotropic meshes after six refinement steps for the DG(1) method using a derivative indicator based on the Mach number.

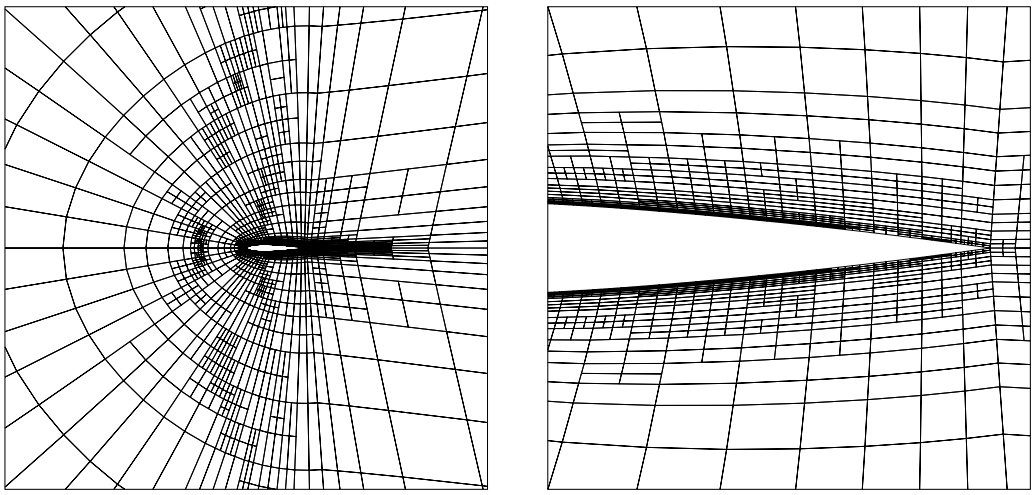
of a stronger refinement of the leading and trailing edge and an additional refinement along a line starting from the point, where the shock touches the airfoil. This line represents a characteristic of the solution to the adjoint problem. This refinement underlines the importance of the exact position of the shock. Clearly, this is more important than a better refinement of the shock, as the mesh obtained with this error indicator yields a target functional error, that is reduced by a factor of approximately 2.5 compared to the corresponding mesh using residual based error indicators and a comparable number of elements. As the shock is reasonably aligned with the mesh, at least in the upper part, anisotropic elements can be used effectively in both cases.

7.5.3. Comparison for test case C

The supersonic viscous flow considered in test case C features both a detached bow shock in front of the airfoil and boundary layers at the surface of the airfoil. Because of these strong anisotropic features a considerable reduction of elements could be expected for anisotropic refinement. However, as the shock is poorly aligned with the mesh in most parts of the domain, these expectations have to be reduced.
Examples of the adapted meshes are given in Figure 7.17, which shows the meshes after five anisotropic refinement steps with the derivative indicator, both for residual based and adjoint based error indicators. Like in test case A, an anisotropic derivative indicator based on the conservative variables is used.
Using residual based error indicators the refinement concentrates on the complete bow shock. As this shock is not well aligned with the initial mesh, anisotropic refinement cannot be used efficiently, although some anisotropic elements appear, mainly induced by mesh smoothing resulting from hanging nodes of the multiply refined elements near the shock. Additional refinement takes place in the wake, where anisotropic refinement



(a) residual based error indicators



(b) adjoint based error indicators

Figure 7.17.: Test case C: Adapted anisotropic meshes after five refinement steps for the DG(1) method using a derivative indicator based on the conservative variables: distant view (left) and zoom of the trailing edge (right).

is quite effective. However, the boundary layer is hardly refined up to this refinement step. This situation changes drastically, if adjoint based error indicators are used, which leads to a refinement mainly in the boundary layer. The bow shock and the wake are only refined in the vicinity of the airfoil, where the flow strongly influences the target functional value. Refining the shock and wake far away from the airfoil seems not to be as efficient as refining the boundary layer. As the latter is well aligned with the mesh, anisotropic refinement is quite effective. The aspect ratio of the boundary layer elements might be expected to be higher. However, the Reynolds number is low ($Re = 1000$) which results in quite a weak boundary layer. Furthermore, the aspect ratio of some elements can still increase after additional refinement steps.
Figure 7.18 shows a comparison between isotropic refinement and anisotropic refinement based on the jump indicator as well as a derivative indicator. All computations are based on the DG(1) method and start on the coarse initial mesh. Both, residual based and adjoint based error indicators are used.

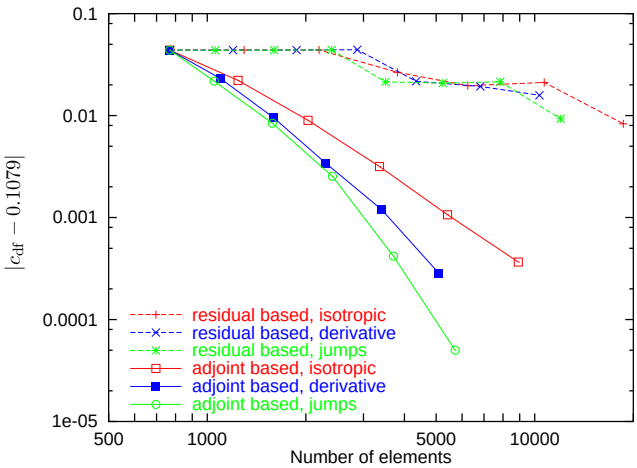


Figure 7.18.: Test case C: Convergence of the error in c_{df} for the DG(1) method using residual and adjoint based error indicators combined with anisotropic jump and derivative indicators.

The advantage of adjoint based error indicators, which can be seen from the adapted meshes already, is reflected in the convergence plot. For the first few refinement steps, the target functional error does not decrease, if residual based indicators are used. The refinement of the complete bow shock does not contribute noticably to an improved target functional value. Therefore, the error does not decrease until the vicinity of the airfoil is refined after some additional steps of the adaptive algorithm. As anisotropic elements are not effective in the bow shock due to little alignment with the initial mesh, no significant improvement can be achieved using anisotropic refinement.
Considering the convergence achieved with adjoint based error indicators, using an anisotropic refinement strategy a considerable number of elements can be saved on the finest mesh, over 40 % of the elements in case of the derivative indicator and even more than 50 % in case of the jump indicator. Furthermore, the achieved accuracy is very similar for isotropic refinement and anisotropic refinement based on the derivative indicator, whereas the accuracy reached after a given refinement step even increases for the jump indicator.

7.5.4. Indicator recommendation

The adaptive anisotropic mesh refinement algorithm described in this work and used for the test cases relies on two different types of indicators: error indicators to select the elements to be refined and anisotropic indicators to decide upon a refinement case for these elements.

The numerical examples presented in the previous sections show that adjoint based error indicators are clearly advantageous. The discrete approximation on the obtained adapted meshes shows a significantly lower target functional error as compared to the results achieved with residual based error indiactors. This behavior, which is especially pronounced in test cases A and C, justifies the additional effort of solving the adjoint problem. Similar results have been shown in [9, 15].

Concerning anisotropic indictors, the results show that both types of the developed indicators can be used successfully to govern an anisotropic refinement process. The reduction in the number of elements achieved was about 40 % in most cases. Generally, the derivative indicator seems to be slightly advantageous for subsonic flow simulations, whereas the jump indicator proves advantageous for trans- and especially supersonic cases. It performs only slightly worse in boundary layers but apparently considerably better in the vicinity of shock layers. This behavior is caused by the non-smooth features of the solution at shocks, which violate the basic assumption included in the derivative indicator.

A derivative indicator based on the individual evaluation of the conservative variables should be used for subsonic viscous flows, whereas the jump indicator is advantageous for trans- and supersonic flows. For inviscid flows, anisotropic features can be found mainly in shocks, thus the jump indicator seems advantageous. Alternatively, a derivative indicator based on the Mach number can be applied.

In some cases, the accuracy achieved after a given number of refinement steps reduces slightly compared to the isotropic refinement case. This might lead to an increased number of refinement steps before the adaptive process can be stopped. However, this effect can be reduced by an increase of the refinement fraction f_r.

The threshold values have to be chosen carefully. In order to prevent anisotropic refinement in inappropriate situations, θ, θ_r and $\tilde{\theta}_r$ have to be chosen large enough, whereas θ_α has to be chosen small enough. On the other hand, too restrictive values reduce the possible saving effect. The values 3.0 for all threshold ratios and 15° for the threshold angle seem to be adequate for several test cases.

8. Conclusion and outlook

The knowledge of flow phenomena and flow-induced forces is vital to many fields of science and engineering. Typical aerodynamic applications ask for the value of certain target functionals, mainly force and momentum coefficients, combined with a certain accuracy requirement. As physical measurements and analytical solutions are not available in most cases, efficient numerical approximations are required.
Discontinuous Galerkin methods offer stable and locally conservative approximations. Combined with adaptive mesh refinement, efficient techniques can be created. In order to drive the adaptive mesh refinement algorithm, error indicators have to be evaluated in addition to the solution of the primal problem. Residual based indicators offer reliable information on errors in the flow solution. Dual weighted residual error indicators additionally offer enhanced information about the location of elements contributing largely to the error of a selected target functional, thereby enabling a significantly improved refinement, resulting in meshes specifically tailored to the effective computation of the desired target functional. A minor drawback of using such a goal oriented adaptive refinement is the additional effort needed to obtain the solution of the adjoint problem. However, this solution and the evaluated error indicators allow a sharp estimation of the target functional error, thereby enabling the adaptive refinement to stop when the desired accuracy is reached.
The efficient numerical approximation of flow fields and derived target functionals requires appropriate meshes. Adaptive refinement is only one key component to obtain optimal meshes. An additional gain can be achieved, if the mesh refinement accounts for anisotropic features of the solution. At interior layers like shocks and boundary layers the flow features change rapidly in a direction orthogonal to the layer whereas the change is much smaller parallel to the layer. Therefore, refinement orthogonal to these layers is much more efficient than refinement in the direction of the layers. A very effective refinement strategy can be achieved, if the selection of elements to be refined based on an accurate error estimation is combined with a decision on an appropriate anisotropic refinement case. This latter decision is based on an anisotropic indicator. In this work, two distinct concepts have been presented and evaluated. The first indicator is based on the jumps of the solution over element faces that occur in DG methods. A large average jump in one direction indicates a large error in this direction. The second indicator is based on the estimation of the interpolation error. In sufficiently smooth regions of the flow the $(p+1)$th derivative of the approximation can be used to approximate the interpolation error of a DG(p) method. If the derivative is especially large in one direction, refinement along that direction will be more efficient than refinement in other directions. The described anisotropic indicators have been applied to several simple test cases for compressible flows over a NACA0012 airfoil. For viscous subsonic flows, a derivative indicator based on the conservative variables proves advantageous, but the jump indicator can also be applied successfully. In contrast to that, for inviscid flows and supersonic cases the jump indicator is more effcient. For inviscid flows a derivative indicator based on the Mach number instead of the conservative variables leads to an improved reduction

in the number of elements needed to obtain a given accuracy. In most cases about 40 % of the elements can be saved on the finest mesh. As the numerical effort generally rises at least linearly with the number of unknowns in the discrete system, this constitutes a considerable reduction in computational time as well as memory requirements. The evaluation of both indicators is very cheap and thus the gain in performance is not reduced by an additional effort.
The developed indicators can be applied with similar success to higher order DG methods. The combination with an error estimation technique has been tested for residual based and adjoint based error estimation. Whereas the reduction in the number of cells needed is slightly higher in case of a residual based error estimation, the overall advantage of adjoint based error estimation can clearly be seen. Even considering the additional effort of solving the adjoint problem the convergence of target functional errors is much better using a goal oriented refinement driven by adjoint based error indicators.
The presented results are promising and allow more efficient numerical approximations to a certain class of flow problems. The applicability is limited, however, with respect to the problem, as well as with respect to the numerical method. Concerning restrictions of the physical problem, only two–dimensional laminar flows can be approximated so far. From the numerical point of view the restrictions are given by the application of pure h–refinement and by the use of unit square reference elements. In order to enable the efficient evaluation of realistic, complex flow problems, several aspects have to be taken care of. Possible future work includes:

- **Extension to three dimensions**. As the concepts described in this work are generally independent of the space dimension, the extension to three-dimensional problems is mainly a matter of implementational effort. It should be expected that the possible savings increase for three–dimensional problems, as elements will be refined only along one out of three directions near anisotropic features of the solution.

- **Extension to turbulent flows**. The scope of this work has been limited to laminar flows mainly because of the restricted abilities of the flow solver used as basis for the calculations. The presented concepts can be applied immediately to turbulent flows. The only aspect to be considered is the possible inclusion (or exclusion) of the additional variables of the selected turbulence model into the evaluation of anisotropic indicators. Some numerical examples should suffice to make this decision. Again, the possible savings will probably increase for higher Reynolds numbers as the boundary layer is much stronger in these cases.

- **Extension to hp–refinement** in order to enable optimal convergence rates. Probably the setting of refinement indicators will be extended to a third step for this procedure. After evaluating a general error indicator and selecting the elements to be refined, in a second step these elements are evaluated with respect to anisotropic features. Additionally, in a third step an indicator deciding between h– and p–refinement would have to be evaluated.

- **Extension to hybrid meshes**. It is very difficult to generate high quality meshes for complex geometries consisting only of quadrilateral or hexahedral elements, respectively. Therefore, it is desirable to use hybrid meshes, which additionally allow triangular elements in two–dimensional domains or tetrahedra, pyramids and

prisms in three dimensions. Solving flow problems on hybrid meshes is mainly dependent on the implementational effort. However, adaptive refinement, especially anisotropic refinement, presents some interesting and unanswered questions. The indicators presented in this work rely on the fact, that there are two faces opposite each other and d orthogonal characteristic directions of the element, at least on the reference element. In general, this is not the case. Furthermore, refining an element it may be possible to choose between different element types of the children. It is possible, for example, to refine a quadrilateral into two or four triangles instead of two or four quadrilaterals. Choosing the best refinement strategy, especially in three dimensions, will be a major task and will require the development of elaborate anisotropic indicators.

Bibliography

[1] M. J. Aftosmis and N. Kroll. A quadrilateral based second–order TVD method for unstructured adaptive methods. AIAA 91-0124, 1991.

[2] W. Bangerth, R. Hartmann, and G. Kanschat. `deal.II` *Differential Equations Analysis Library, Technical Reference*. http://www.dealii.org/, 5.2 edition, 2005. First edition 1999.

[3] M. J. Castro-Díaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaption for flow simulations. *International Journal for Numerical Methods in Fluids*, 25:475–491, 1997.

[4] L. Formaggia, S. Micheletti, and S. Perotto. Anisotropic mesh adaptation in computational fluid dynamics: Application to the advection-diffusion-reaction and the Stokes problems. *Applied Numerical Mathematics*, 51:511–533, 2004.

[5] L. Formaggia and S. Perotto. New anisotropic a priori error estimates. *Numerische Mathematik*, 89:641–667, 2001.

[6] P. J. Frey and F. Alauzet. Anisotropic mesh adaption for transient flow simulations. In *12th International Meshing Roundtable, Sandia National Laboratories*, pages 335–348, 2003.

[7] P. J. Frey and F. Alauzet. Anisotropic mesh adaptation for CFD computations. *Comput. Methods Appl. Mech. Engrg.*, 194:5068–5082, 2005.

[8] E. H. Georgoulis. *Discontinuous Galerkin methods on shape–regular and anisotropic meshes*. PhD thesis, University of Oxford, 2003.

[9] R. Hartmann. Discontinuous Galerkin methods for compressible flows: higher order accuracy, error estimation and adaptivity. In H. Deconinck and M. Ricchiuto, editors, *VKI LS 2006-01: CFD-Higher Order Discretization Methods, November 14-18, 2005, Rhode Saint Genese, Belgium*. Von Karman Institute for Fluid Dynamics, 2005.

[10] R. Hartmann. Adaptive discontinuous Galerkin methods with shock-capturing for the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 51(9–10):1131–1156, 2006.

[11] R. Hartmann. Derivation of an adjoint consistent discontinuous Galerkin discretization of the compressible Euler equations. In G. Lube and G. Rapin, editors, *Int. Conference on Boundary and Interior Layers, BAIL2006*, 2006.

[12] R. Hartmann. Adjoint consistency analysis of discontinuous Galerkin discretizations. *SIAM J. Numer. Anal., submitted*.

[13] R. Hartmann and P. Houston. Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations. *Journal of Computational Physics*, 183:508–532, 2002.

[14] R. Hartmann and P. Houston. Symmetric interior penalty DG methods for the compressible Navier–Stokes equations I: Method formulation. *International Journal of Numerical Analysis and Modeling*, 3(1):1–20, 2006.

[15] R. Hartmann and P. Houston. Symmetric interior penalty DG methods for the compressible Navier–Stokes equations II: Goal–oriented a posteriori error estimation. *International Journal of Numerical Analysis and Modeling*, 3(2):141–162, 2006.

[16] R. Hartmann *et al.*. *PADGE, Technical Reference.* DLR Braunschweig, 2006.

[17] P. Houston, E. H. Georgoulis, and E. Hall. Adaptivity and a posteriori error estimation for DG methods on anisotropic meshes. In G. Lube and G. Rapin, editors, *Int. Conference on Boundary and Interior Layers, BAIL2006*, 2006.

[18] W. Huang. Metric tensors for anisotropic mesh generation. *Journal of Computational Physics*, 204:633–665, 2005.

[19] O. Sahni, J. Müller, K. E. Jansen, M. S. Shepard, and C. A. Taylor. Efficient anisotropic adaptive discretization of the cardiovascular system. *Comput. Methods Appl. Mech. Engrg.*, 195:5634–5655, 2006.

[20] S. Sun and M. F. Wheeler. Anisotropic and dynamic mesh adaption for discontinuous Galerkin methods applied to reactive transport. *Comput. Methods Appl. Mech Engrg.*, 195:3382–3405, 2006.

[21] D. A. Venditti and D. L. Darmofal. Anisotropic grid adaption for functional outputs: application to two–dimensional viscous flows. *Journal of Computational Physics*, 187:22–46, 2003.

A. Implementational issues

The obvious implementational effort for anisotropic refinement is limited to simple functions, which are capable of splitting an element according to a given refinement case and storing the obtained geometrical and topological data. There are, however, several additional aspects that need to be considered. Some of these aspects and the suggested solutions of arising difficulties are directly connected to the code used to implement the numerical method. The following remarks will therefore be concerned with the issues that arised in this work using a specific code. Other implementations might not have to consider some of the aspects, whereas other difficulties might arise.

A.1. Status of `deal.II` and data storage structure

The practical part of this work in connected to the DG flow solver `PADGE` [16] currently developed at the DLR (Deutsches Zentrum für Luft- und Raumfahrt) Braunschweig, which is based on the `C++` Finite Element library `deal.II` [2]. For general information or details of specific aspects, the freely available documentation is recommended. Some important aspects, however, that are directly connected to this work, will be discussed in the following.

So far, isotropic refinement is hard-coded in `deal.II`. The elements, which are always quadrilaterals in two dimensions and hexahedrals in three dimensions, can only be refined into four or eight children, respectively. Furthermore, the number of hanging nodes is limited to one hanging node on each mesh edge on refined meshes, whereas it must be zero on all edges for the initial mesh. The elements are organized in levels, where the first level represents the initial mesh. Further levels are filled with refined elements only. An element belonging to the second level is a child of an element belonging to the initial mesh, whereas this child's own children would belong to the third level and so forth. The current mesh after a refinement step consists of all elements on the highest level plus a combination of elements from lower levels, which together cover the whole domain. All elements without children belong to the refined mesh as well.

Along with the elements, faces and edges are stored in the same structure of levels. As the initial mesh is free of hanging nodes, each face and edge is uniquely bounding elements of one level only, if isotropic refinement is used. Vertices can belong to elements of different levels, however, and are stored globally for a triangulation.

If anisotropic refinement is used, edges and faces can no longer be uniquely associated with one level. This is obvious from the very simple example shown in Figure A.1.

This example shows two elements κ_1 and κ_2, both belonging to level n, the left of which is refined anisotropically into the elements κ_{1_1} and κ_{1_2}, both belonging to level $n+1$. The line l is the common edge and face of elements κ_1 and κ_2 and thus belongs to level n. However, l is also an edge and face of element κ_{1_2}, and, therefore, belongs to level $n+1$. Refining κ_{1_2} anisotropically again, the number of levels line l would have to belong to increases. One simple solution would be to duplicate line l to line $\bar{l}$ on all levels it is needed on.

若采用各向同性加密，则每个face只属于一个level.

l只属于level$_1$.

细分后K_{1_2}的边l'是新生成的. l'只属于level$_2$.

若采各向异性加密，情况就接不上

细分后K_{1_2}的边l'仍是原来的边l
故l既属于level$_1$，又属于level$_2$.

(a) original state on level n

(b) anisotropically refined state on levels n and $n+1$

Figure A.1.: Example of a refinement case with a line (edge, face) belonging to elements of more than one level.

However, simply duplicating edges and faces imposes a strong restriction on the allowed refinement cases of neighboring elements. Consider, for example, element κ_2 being refined anisotropically as shown in Figure A.2.

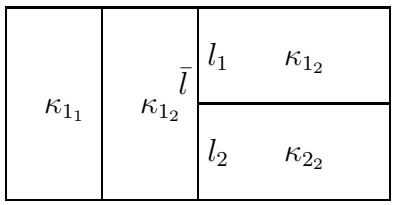


Figure A.2.: Example of a refinement case resulting in corrupted data.

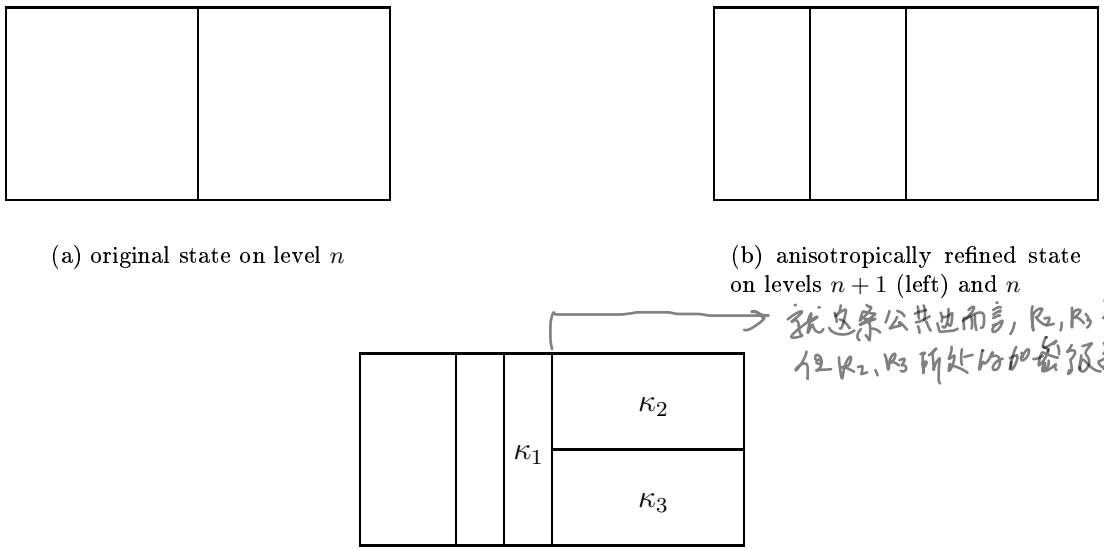
Line l has to be split into two children l_1 and l_2 in order to bound the newly created elements κ_{2_1} and κ_{2_2}. However, this only applies to the (original) line l on level n, whereas the duplicated line $\bar{l}$ on level $n+1$ is unchanged. After the refinement, line l has children, whereas $\bar{l}$ has none, although both correspond to the same line in real space. Therefore this refinement case cannot be allowed in order not to corrupt data integrity. 不能破坏数据完整性 These observations give rise to the need to detach edges and faces from the level-based data structure of elements and store them globally like the vertices. The object oriented structure of `deal.II` makes it necessary to modify not only the storage classes themselves but also the accessor and iterator classes used for access to the data of any object of a triangulation. It is *not* necessary, however, to modify those parts of the code actually using edges and faces, as the access to the required data is never done directly, but only by means of the accessors and iterators mentioned above (the *data encapsulation* concept often used in object oriented programming).
As the data storage of degrees of freedom mirrors the structure used for the storage of triangulation data, the changes described above have to be applied in an analog way to the degrees of freedom storage and accessor classes.

A.2. Connectivity

Based on the restriction to a maximum of one hanging node on an edge, an element can only be in four states concerning its neighbors at a given face:

1. The element is at the boundary, thus having no neighbor element (at this face).
2. The element is neither coarser nor finer than its neighbor element.
3. The element is exactly one step finer than the neighbor element.
4. The element is exactly one step coarser than the neighbor elements.

The first case is trivial and needs no special treatment for anisotropic refinement. In the second case, the level of the element and its neighbor coincide in the case of pure isotropic refinement. If the element is one step finer than the neighbor, the element's level is higher by one as compared to the neighbor's level, whereas it is lower by one if the element is coarser than the neighbor. This unique relationship between the refinement situation of neighboring elements and the levels these elements belong to has motivated the use of level–differences as criterion for the local refinement situation. As some procedures are dependent on this situation, the decision between different cases is made based on the levels of the involved elements.
This cannot be done in the case of anisotropic refinement, as the situation can be different at each face of an element and does not depend on the level anymore. Figure A.3 shows an example refinement history, in which element κ_1 is coarser than elements κ_2 and κ_3 along the common face. κ_1 is on level $n+2$, however, whereas the more refined elements κ_2 and κ_3 are both on level $n+1$, resulting in the more refined elements being on a coarser level.



(a) original state on level n

(b) anisotropically refined state on levels $n+1$ (left) and n

就这条公共边而言，κ_2，κ_3 是比 κ_1 更细化的。但 κ_2、κ_3 所处的单元级别更低一些。

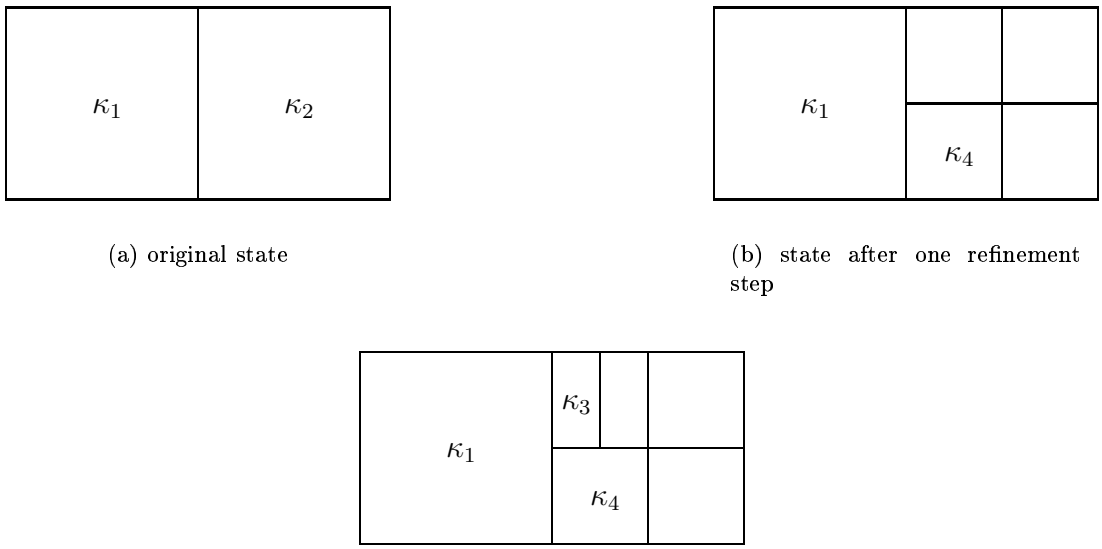
(c) anisotropically refined state on levels $n+2$ (middle) and $n+1$

Figure A.3.: Example of a refinement case with the coarser element being on a higher level.

This example illustrates that any decision on the refinement situation between two neighboring elements cannot be made based on the levels of the involved elements, but has

也就是说所谓"neighbor"的概念是它必须比
当前网格更稀疏才算邻居.

to be done in a different way. In `deal.II`, the stored neighbor of an element at a given face is never more refined. If there are refined neighboring elements at this face, then they are the children of the stored neighbor. This makes it easy to identify case 4 of the above mentioned four cases: if the stored neighbor has children, than the real neighbors are more refined than the element under consideration. As case 1 is obvious as well, it remains to decide between cases 2 and 3. In order to make this decision, the faces of the involved elements can be considered. If the two elements share one face, neither one is more refined or coarser than the other, resulting in case 2. In all remaining situations the neighbor is coarser than the current element and case 3 is recovered.
It has to be noted, that there are further, less obvious changes concerning the neighborship situation of a given element, especially in the case of an element with more refined neighbors. It is possible in the case of anisotropic refinement, that the element really neighboring the element under consideration is not a direct child of the stored neighbor, but a grandchild, grand-grandchild etc. An example of this situation is given in Figure A.4, in which element κ_1 is neighbored by elements κ_3 and κ_4, where κ_3 is a grandchild of the original element κ_2, the stored neighbor of element κ_1.



(a) original state

(b) state after one refinement step

(c) state after two refinement steps

Figure A.4.: Example of a refinement case with a refined neighbor, which is not the stored neighbor's (direct) child.

The possibility of stored neighbor's children, which are refined (several times) in an anisotropic way that does not refine the face bounding the element presently under consideration, has to be taken care of in the search for the real neighbors of a given element.

A.3. Prolongation and restriction operators

As described in Section 4.1, h–refinement yields embedded function spaces for the discrete solution. This enables an efficient and lossless interpolation from the coarse mesh to the refined one. This interpolation for the whole mesh is based on interpolating the approximated solution from the mother element to its children for all refined elements[11]. Similar to that, the solution on a refined mesh can be interpolated to a coarser mesh. However, this process is not lossless.
The interpolation can be done separately for each element under consideration. For elements in the interior of the domain the mapping between unit and real element is only scaled by a factor of 2 between an element and its children, otherwise it stays the same. That is why the interpolation rule can be defined on the unit element, enabling to use the same rule for all elements.
Degrees of freedom are merely factors to basis functions ϕ. Any discrete function $\mathbf{u}$ is a linear combination of degrees of freedom $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n)^T$ and basis functions $\phi_i, i = 1, \ldots, n$, combined in the vector of basis functions $\underline{\phi}$:

$$\mathbf{u} = \hat{\mathbf{u}}^T \, \underline{\phi} = \sum_{i=1}^{n} \hat{u}_i \, \phi_i. \tag{A.1}$$

Prolongation

In the environment of multigrid algorithms the interpolation from a mother element to refined children is known as *prolongation*.
A unit square or unit cube in two or three dimensions, respectively, is considered, containing one finite element: the mother element. This one–element mesh is refined according to the refinement case under consideration. For each child element κ_{c_i}, $i = 1, \ldots, n_{\text{children}}$, where n_{children} is the number of children an element has for the given refinement case, an individual matrix $P^{(i)}$ is defined, such that the mother's degree of freedom values $\hat{\mathbf{u}}^{(m)}$ and the degree of freedom values $\hat{\mathbf{u}}^{(c_i)}$ of child i are connected through

$$\hat{\mathbf{u}}^{(c_i)} = P^{(i)} \, \hat{\mathbf{u}}^{(m)}, \quad i = 1, \ldots, n_{\text{children}}. \tag{A.2}$$

As only pure h–refinement is considered, $P^{(i)}$ is a $n \times n$ square matrix, where n is the number of basis functions per element, which is identical to the number of degrees of freedom per element in the case of DG methods.
In order to compute the prolongation matrices $P^{(i)}$, conditions have to be imposed. As the interpolation from coarse to fine elements can be done in a lossless way, the following condition is obvious:

$$\mathbf{u}^{(m)}(\mathbf{x}) = \mathbf{u}^{(c_i)}(\mathbf{x}) \quad \forall \mathbf{x} \in \kappa_{c_i}, \quad i = 1, \ldots, n_{\text{children}}, \tag{A.3}$$

where the superscripts $^{(m)}$ and $^{(c_i)}$ refer to coarse and fine mesh values of mother and child element values, respectively. Denoting the mother element basis functions with Φ and the child's basis functions with ϕ, Equations (A.2) and (A.3) can be combined, using

[11] On curved boundaries the boundary description and, thereby, the mapping from reference to real space element changes after the refinement. Thus, the interpolation is not completely lossless at the boundary.

the summation convention, to yield

$$\hat{u}_j^{(m)}\,\Phi_j(\mathbf{x}) = \hat{u}_k^{(c_i)}\,\phi_k(\mathbf{x}) = P_{kj}^{(i)}\,\hat{u}_j^{(m)}\,\phi_k(\mathbf{x}) \quad \forall \mathbf{x} \in \kappa_{c_i}, \quad i = 1, \ldots, n_{\text{children}}, \tag{A.4}$$

where j and k run from 1 to n over all basis functions of the mother and child cell, respectively. Equation (A.4) must be valid, independent of the values of $\hat{u}_j^{(m)}$. Therefore, the equation has to be fulfilled for each individual component, resulting in the set of equations

$$\Phi_j(\mathbf{x}) = P_{kj}^{(i)}\,\phi_k(\mathbf{x}) \quad \forall \mathbf{x} \in \kappa_{c_i}, \quad j = 1, \ldots, n, \quad i = 1, \ldots, n_{\text{children}}. \tag{A.5}$$

For fixed j, selecting a sufficiently high number of control points $\mathbf{x}_c$ inside κ_{c_i} and demanding that Equation (A.5) is fulfilled for these points, an overdeterminded linear system of equations is obtained, which can be solved in a least squares sense for the values $P_{kj}^{(i)}$, $k = 1, \ldots, n$. The points $\mathbf{x}_c$ have to be chosen such that they yield n linearly independent equations in (A.5). In case of basis functions, which are based on support points, like Lagrangian interpolation polynomials, the $\mathbf{x}_c$ can simply be chosen to be the support points. This way, only n control points are used, uniquely defining all desired components $P_{kj}^{(i)}$. However, in order to reduce the influence of numerical errors, it is advantageous to employ the suggested least squares problem. One such problem has to be solved for each basis function of the mother element. This yields one interpolation operator $P^{(i)}$ for one child of a specific refinement case. In turn, one such set of least squares problems has to be solved for each child occuring in each refinement case. These problems have to be solved only once, if the obtained matrices are stored in order to use them on the mesh elements in real space. There, the degrees of freedom on child i can be interpolated from the degree of freedom values on the mother element by applying the appropriate prolongation matrix for the refinement case at hand.

Restriction

The process of interpolating a function or the corresponding degrees of freedom from all children of an element to the mother element is known as *restriction*. It is used in multigrid algorithms as well as in the coarsening of elements in adaptive refinement processes. An expression of the type

$$\hat{\mathbf{u}}^{(m)} = \sum_{i=0}^{n_{\text{children}}} R^{(i)}\,\hat{\mathbf{u}}^{(c_i)} \tag{A.6}$$

is seeked, where $R^{(i)}$, $i = 1, \ldots, n_{\text{children}}$, represents the restriction matrix corresponding to the child i of the mother element.

As this interpolation is not lossless, there are several possible definitions of this interpolation. One of the possibilities is given by the L^2–projection, which requires that integrating a function $\mathbf{u}$ over the element κ_{c_i}, weighted by a test function $\mathbf{v}$, yields the same value for both the coarse and the fine mesh functions $\mathbf{u}^{(m)}$ and $\mathbf{u}^{(c_i)}$:

$$\int_{\kappa_{c_i}} \mathbf{u}^{(m)} \cdot \mathbf{v}\,\mathrm{d}x = \int_{\kappa_{c_i}} \mathbf{u}^{(c_i)} \cdot \mathbf{v}\,\mathrm{d}x, \quad i = 1, \ldots, n_{\text{children}}. \tag{A.7}$$

Like in the case of prolongation, the functions $\mathbf{u}^{(m)}$ and $\mathbf{u}^{(c_i)}$ in Equation (A.7) are re-

placed by their respective representations of degree of freedom values and basis functions. Afterwards, the restriction operation defined in Equation (A.6) is introduced. This yields

$$\int_{\kappa_{c_i}} \left(\hat{u}_j^{(c_i)}\,\phi_j\right)\cdot\mathbf{v}\,\mathrm{d}x = \int_{\kappa_{c_i}} (\hat{u}_k^m\,\Phi_k)\cdot\mathbf{v}\,\mathrm{d}x = \int_{\kappa_{c_i}} \left(R_{kj}^{(i)}\,\hat{u}_j^{(c_i)}\,\Phi_k\right)\cdot\mathbf{v}\,\mathrm{d}x, \quad i = 1,\ldots,n_{\text{children}}. \tag{A.8}$$

Again, this Equation has to be true for all discrete functions $\mathbf{u}$, resulting in n equations, one per component of $\hat{u}^{(c_i)}$. Choosing the coarse element basis functions Φ_l, $l = 1,\ldots,n$, as test functions $\mathbf{v}$ and exchanging the order of summation and integration yields the following system of linear equations:

$$\int_{\kappa_{c_i}} \phi_j \cdot \Phi_l\,\mathrm{d}x = R_{kj}^{(i)} \int_{\kappa_{c_i}} \Phi_k \cdot \Phi_l\,\mathrm{d}x \quad j,l = 1,\ldots,n, \quad i = 1,\ldots,n_{\text{children}}, \tag{A.9}$$

which can be used to determine all components of the restriction operator $R^{(i)}$ for child i concerning a specific refinement case. Again, this procedure has to be repeated for all children of all possible refinement cases.

A.4. Mesh smoothing

Having evaluated all error indicators and optionally anisotropic indicators, some of the active elements are flagged for refinement, others are flagged for coarsening. Before the actual refinement and coarsening can take place, a mesh smoothing algorithm has to be applied. This algorithm is based on two distinct ideas:

1. In order to end up with a valid state of the mesh after the refinement, the limitation of the number of hanging nodes has to be enforced. In general, the condition of only one hanging node at each edge will not be fulfilled automatically. It is, therefore, necessary to unset refinement flags on elements already flagged for refinement, to unset coarsening flags, or to set additional refinement flags on other elements. As the purpose of an adaptive refinement step is to increase the approximation accuracy, only the latter two strategies seem reasonable.
   Furthermore, coarsening may only take place on elements of which all children are flagged for coarsening. If one of the children is not flagged, the flags of all other children are erased. This again favors finer elements over coarser ones in order to obtain improved approximation accuracy.

2. Apart from these necessary checks of refinement and coarsening flags it might be desirable to apply additional smoothing steps, which result in a somewhat more regular refined mesh. Several distinct algorithms can be employed, ensuring, for example, that an element is refined if all or most of its neighbors are refined already or will be after the current refinement step. This avoids unrefined islands, which might reduce the approximation accuracy not only on the element under consideration, but also on the surrounding patch of elements. However, this effect is more important for continuous Finite Elements than for DG methods. For a description of other useful smoothing algorithms implemented in `deal.II` refer to [2].

Individual algorithms have to be applied to meet the different requirements arising from the obligatory tasks as well as the additional smoothing requests. As these algorithms can

interact in a very complicated and unpredictable way, the algorithms have to be applied sequentially, with the most important ones at the end of the sequence to ensure, that their work is not undone by following actions. If no flags have been changed in a complete run of this sequence, the process can be terminated, otherwise the whole sequence has to be repeated. All algorithms use loops over all elements. Several algorithms need several passes of this loop to ensure, that the underlying requirement is fulfilled for all elements, others can ensure this for a single run of the loop over all elements, if this loop is performed in a special way, for example treating the finest elements first, going to coarser elements afterwards.
Concerning anisotropic refinement, there are some important differences in the algorithms as compared to pure isotropic refinement.

Local refinement situation

It is obvious from the reasoning in Section A.2 that any decision on the local refinement situation, including the information, which of two neighboring elements is coarser, cannot be based upon the levels of the involved elements in case of anisotropic refinement. The conditions presented in Section A.2 have to be implemented in all algorithms as a replacement of level–based decisions.

Loops over all elements

As mentioned above, it is known that some algorithms terminate after one single pass of the loop over all elements, if they are performed running from finer to coarser elements. This is not possible in the case of anisotropic elements. Firstly, it is not possible to simply run backwards over the levels of the triangulation, as elements on a lower level may be finer than neighboring elements along one face. Secondly, elements may be very fine along two opposite faces and quite coarse along the other ones. Because of these difficulties it is not possible to ensure that the algorithms under consideration terminate after a single loop. Performing only one pass of the loop regardless of this fact might slow down the convergence of the sequential run over all algorithms. Performing an inner loop, terminated if no flags are reset during a whole pass over all elements, could be more effective[12].

Anisotropic smoothing

Refining one element twice and not refining its neighbor would result in too many hanging nodes, thus the neighboring element has to be refined, too. However, that particular element was not flagged for refinement by the error indicators, which suggests that the refinement would probably not be necessary in order to improve the approximation accuracy, at least it does not have a high priority. It will be enough in most cases to do this additional refinement anisotropically, only splitting those edges which are required to be split due to the number of hanging nodes.
This technique offers another advantage. Consider, for examples, elements in a boundary layer, which are refined anisotropically, resulting in a high aspect ratio. If one of the elements is flagged for an additional refinement the neighboring element might also need

[12] As information is passed over at least one level in each run over all elements, the inner loop terminates after at most n_{levels} cycles, where n_{levels} is the number of levels in the triangulation. Thus, the algorithm does *not* show quadratic convergence.

to refine itself due to the restriction of hanging nodes. However, this should clearly be done anisotropically in the boundary layer. Perhaps this element would have been flagged anyway, had the refinement fraction been slightly higher. Refining this element isotropically would result in an avoidable, useless increase in the number of elements.

Refinement situation on faces

Similar to anisotropic smoothing, several optional smoothing algorithms make decisions based on the refinement of the neighboring element. In case of isotropic refinement there is only one refinement property of each element, but in the case of anisotropic refinement, this property is dependent on the face under consideration. Other than in most parts of the code, in mesh smoothing algorithms not only the current refinement situation is important, but also the new situation after the refinement step. If an element is not coarser than its neighbor at the moment, it is an important information, if the neighbor will be refined after the refinement step. However, it is not sufficient to ask, whether the neighbor's refinement flag is set, but rather whether the neighbor will be refined along the face common with the current element. In other words it is important to know, how a specific refinement case will influence the refinement of a specific face of the element under consideration. The same information is needed for edges. Appropriate functions returning this information have to be implemented and used in the smoothing algorithms.

Undefined smoothing algorithm

For the case of isotropic refinement, one of the existing smoothing algorithms limits the difference in refinement not only over faces and edges, but also at vertices. This might be useful, especially for continuous Finite Element methods, in order not to reduce the approximation accuracy. The original algorithm limits the difference of the levels of all elements connected to a given vertex. Using a level–based decision is not possible for anisotropic elements, as explained above. However, in this particular case there is no obvious equivalent formulation for anisotropic elements. It is not only difficult to perform an implementation but it is also not even clear, which local refinement situations should be allowed and which be prevented. Because of these reasons this algorithm is not available in the case of anisotropic refinement.

Computational effort

Basing all decisions concerning the local refinement situation on a simple comparison of the involved element's levels, i.e. two integer values, is a cheap solution. The somewhat more involved comparisons necessary for anisotropic refinement are more expensive. As the additional use of inner loops gives further rise to an increased number of operations, the smoothing of refinement flags is associated with a higher computational effort, if anisotropic refinement is allowed. However, comparing with the overall effort of the numerical method, the time needed to perform the mesh refinement, of which the smoothing algorithm is only one part among others, is negligible. The time needed for the whole program run is dominated by the linear algebra part, i.e. by obtaining the solution to discrete equations.

B. Metric intersection

The anisotropic indicators described in Section 6.2.2 are based on derivatives of the solution. In case of d-linear polynomials as basis for the piecewise approximation of the solution, the tensor of second derivatives, i.e. the Hessian matrix of the solution, has to be evaluated. In order to combine information coming from several components of the solution, these matrices have to be "blended". This can be done using a procedure called *metric intersection*. For the sake of simplicity the following explanations are restricted to the two–dimensional case, but the extension to three dimensions is obvious.
In order to enable the intersection procedure, the Hessian matrices $\mathcal{H}$ are expressed by their eigenvalues λ_i and eigenvectors $\mathbf{v}_i$:

$$\mathcal{H} = \mathcal{R} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \mathcal{R}^{-1}, \tag{B.1}$$

where $\mathcal{R} = (\mathbf{v}_1\ \mathbf{v}_2)$ is the matrix of eigenvectors. Generally, these eigenvalues can be negative, but as the magnitude of the interpolation error only depends on the absolute value of the derivative, it is convenient to use these absolute values. Exchanging the eigenvalues by their absolute values defines the metric $\mathcal{M}$ as

$$\mathcal{M}(\mathcal{H}) = \mathcal{R} \begin{pmatrix} |\lambda_1| & 0 \\ 0 & |\lambda_2| \end{pmatrix} \mathcal{R}^{-1}. \tag{B.2}$$

Such a metric can be obtained for each component of the solution. The question is now, how to create a single metric from these individual ones, which is characteristic of all solution components.
One first step of the following approach is to identify an ellipse as representation of a metric. By

$$\mathbf{x}^T \mathcal{M} \mathbf{x} = 1, \quad \mathbf{x} \in \mathbb{R}^2, \tag{B.3}$$

all points $\mathbf{x}$ are defined, which have a distance of unity from the origin, if this distance is measured with respect to the metric $\mathcal{M}$. These points form the outline of an ellipse having its major axes aligned with the eigenvectors $\mathbf{v}_i$ and semi-axes of length $1/\sqrt{|\lambda_1|}$ and $1/\sqrt{|\lambda_2|}$, respectively. In directions along which the distance to the origin is small, the absolute value of the derivative is large, thus the desired mesh size along this direction is small. Several such ellipses, coming from different solution components, represent different requirements for the mesh. If all these requirements are to be fulfilled, the graphical interpretion is an ellipse, which lies inside all the other ones. In order not to be overrestrictive, the required ellipse is the largest one, that lies completely inside the other ones. Finding such an ellipse and the corresponding metric is not a trivial task. There exist, however, approximations, that have been used with good results for the application at hand. One such approximation is given by Castro-Díaz *et al.* [3] and will be presented in the following.
The problem of finding the intersection of several ellipses is broken down to repeatedly finding the intersection of two ellipses. For this purpose, the two metrics $\mathcal{M}_1$ and $\mathcal{M}_2$

associated to the ellipses $\mathcal{E}_1$ and $\mathcal{E}_2$ are modified to obtain the intermediate metrics $\hat{\mathcal{M}}_1$ and $\hat{\mathcal{M}}_2$. The general idea behind this modification is to use the same eigenvectors for each metric, but to modify the eigenvalue, such that the modified ellipse is restricted along its major axes to the innermost point in that direction: either the one on the original ellipse or the point on the other ellipse, if that point is closer to the origin. In summary, the modified metric $\hat{\mathcal{M}}_1$ is defined by

$$\hat{\mathcal{M}}_1 = \mathcal{R}_1 \begin{pmatrix} \hat{\lambda}_1^{(1)} & 0 \\ 0 & \hat{\lambda}_2^{(1)} \end{pmatrix} \mathcal{R}_1^{-1}, \tag{B.4}$$

where the new eigenvalues $\hat{\lambda}_i^{(1)}$ are given as

$$\hat{\lambda}_i^{(1)} = \max\left(\lambda_i^{(1)}, \mathbf{v}_i^{(1)^T} \mathcal{M}_2 \, \mathbf{v}_i^{(1)}\right). \tag{B.5}$$

The intersection metric $\hat{\mathcal{M}}$ is calculated as the mean value of the intermediate metrics:

$$\hat{\mathcal{M}} = \frac{\hat{\mathcal{M}}_1 + \hat{\mathcal{M}}_2}{2}. \tag{B.6}$$

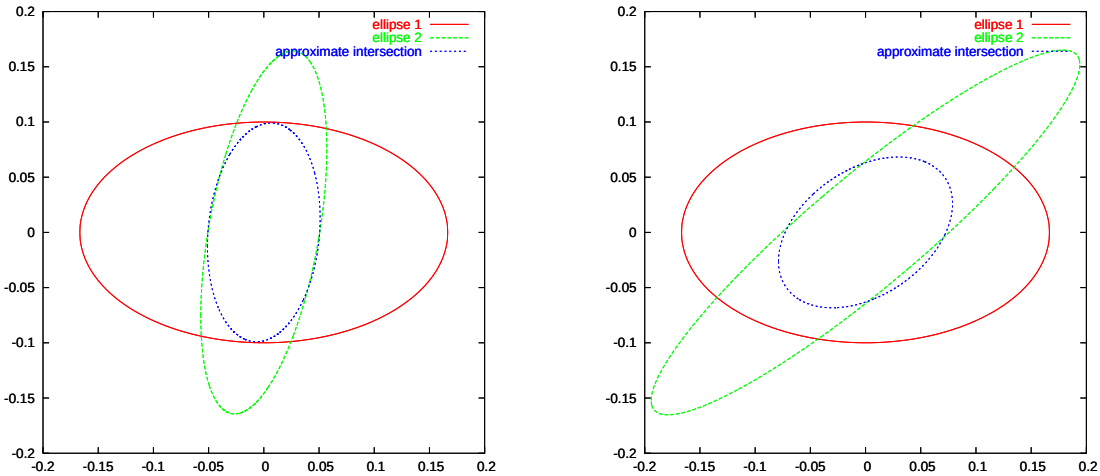
If more than two metrics are to be intersected, the following algorithm is applied:

1. Calculate the intersection of the first two metrics: $\hat{\mathcal{M}} = \text{intersection}\left(\hat{\mathcal{M}}_1, \hat{\mathcal{M}}_2\right)$.
2. For $i = 3, \ldots, n$ calculate the new intersection metric as intersection of metric $\hat{\mathcal{M}}_i$ and the old intersection: $\hat{\mathcal{M}} = \text{intersection}\left(\hat{\mathcal{M}}, \hat{\mathcal{M}}_i\right)$, where n is the number of metrics to be intersected.

The approximation quality can be assessed using the two examples shown in Figure B.1. Whereas in the first case in Figure B.1(a) the result is quite convincing, the second example with different ellipses shows an approximate intersection, which does not seem to be adequate. Varying the shapes and orientations of the ellipses, the quality of the approximation changes from "nearly perfect" to "quite poor".
While the occurance of poor approximations is a fact that would suggest not to use this procedure as a basis for anisotropic refinement indicators, the errors will generally be such, that the resulting approximation has an anisotropic ratio, which is too small as compared to the optimal intersection, i.e. the resulting ellipse is too similar to a circle. In this case it can be expected that some cells, for which anisotropic refinement would be sufficient, are refined isotropically. In the end, this should lead to refined meshes, that use more cells than necessary and are therefore not as efficient as they could be. The occurance of cells, which are refined anisotropically in the wrong direction is not too likely, however. Therefore, it seems possible to use the presented approximate metric intersection in spite of its shortcomings.

Dimension and magnitude of metrics

The components of a vector–valued solution are generally of different dimension and magnitude. The derivatives share these properties. It is therefore difficult to intersect metrics of different components, as the one with the largest values in its respective dimension will likely determine the intersection. In this case, only the anisotropic features



(a) Example with good approximation

(b) Example with poor approximation

Figure B.1.: Approximate intersection of two ellipses representing two metrics.

of this dominant component will be represented by the intersection metric, whereas physically all the components could contribute to the overall anisotropic characteristics. It seems reasonable to use a non-dimensionalized form of the derivatives as basis for the metric construction. One possibility to yield dimension-less derivatives is to devide them by the free-stream values of the respective variables. Another approach is to use local values of the variables instead of constant free-stream values as reference. That way the derivative is representative of the relative instead of the absolute error. Because of this physical interpretation, the latter version is chosen in this work.

Selbständigkeitserklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen verwendet zu haben.

Dresden, den 2. November 2006 _____