Discontinuous Galerkin Finite Element Methods for the Navier–Stokes Equations in Entropy Variable Formulation

Lars Pesch

Colophon

The research presented in this dissertation was carried out at the Numerical Analysis and Computational Mechanics (NACM) group, Department of Applied Mathematics, Faculty of Electrical Engineering, Mathematics and Computer Science of the University of Twente, The Netherlands.

This work has been part of the STW project “Two-phase flows with free surfaces” (TSF.6102) of the FOM/STW-program “Dispersed Multiphase Flows”. The financial support from the Technology Foundation STW, Akzo Nobel, DSM, MARIN and Shell is gratefully acknowledged.

The project has been associated with the J. M. Burgers Research School for Fluid Dynamics.

This thesis was typeset in LaTeX by the author and printed by Wöhrmann Printing Service, Zutphen.

ISBN 978-90-365-2545-9

DISCONTINUOUS GALERKIN FINITE ELEMENT METHODS FOR THE NAVIER–STOKES EQUATIONS IN ENTROPY VARIABLE FORMULATION

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. W. H. M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 28 september 2007 om 15.00 uur

door

Lars Pesch

geboren op 8 juli 1977
te Keulen, Duitsland

Dit proefschrift is goedgekeurd door de promotor,
prof. dr. ir. J. J. W. van der Vegt.

Contents

List of tables

List of figures

List of symbols

The list of symbols is divided into the following categories:

- abbreviations,
- symbols starting with an uppercase Greek letter,
- symbols starting with a lowercase Greek letter,
- symbols starting with an uppercase Roman letter,
- symbols starting with a lowercase Roman letter,
- other symbols.

Where appropriate, a reference to an equation or page number is given.

Abbreviations

ALE	arbitrary Lagrangian–Eulerian
CFL	Courant–Friedrichs–Lewy
EOS	equation of state
EXI	explicit Runge–Kutta method for inviscid flow
EXV	explicit Runge–Kutta method for viscous flow
FE	finite element
FEM	finite element method
IP	interior penalty
LBB	Ladyzhenskaya–Babuška–Brezzi
ODE	ordinary differential equation
OO	object-oriented
OOP	object-oriented programming
PDE	partial differential equation
RK	Runge–Kutta
STL	C++-Standard Template Library
UML	Unified Modeling Language
vdW	van der Waals (equation of state)

Symbols starting with an uppercase Greek letter

Δt_n	n^{th} time step, p. 50
$\Delta\tau$	pseudo-time step, p. 84
Φ_t^n	space mapping at time t within space-time slab n, p. 40
Ψ	extensive quantity, p. 12
Ω	space domain, p. 39

Symbols starting with a lowercase Greek letter

α	specific volume, $\alpha = 1/\rho$, p. 12
α_{c}	critical specific volume of a vdW gas, p. 24
α_p	isobaric expansion coefficient, p. 21
β_T	isothermal compressibility, p. 21
γ	adiabatic coefficient, p. 23
$\delta(\mathcal{S})$	penalization parameter of the IP method on the face $\mathcal{S}$, p. 76
δ_{ij}	Kronecker delta
$\delta_{\Delta\tau}$	pseudo-time step von Neumann number, p. 86
ϵ	artificial compressibility parameter, p. 87
η	dynamic viscosity or shear viscosity, p. 16
$\theta_{\tilde{\imath}\tilde{\jmath}}$	auxiliary variable for the viscous fluxes in the first order system form of the Navier–Stokes equations, p. 67
$\hat{\theta}_{\tilde{\imath}\tilde{\jmath}}$	numerical flux function for the viscous fluxes $\theta_{\tilde{\imath}\tilde{\jmath}}$, p. 76
κ	thermal conductivity, p. 17
λ	Chapter 2: dilatational viscosity or bulk viscosity, Chapter 5: ratio of the pseudo-time step and physical time step
$\lambda^{n,j}$	j^{th} time basis function in slab n, p. 48
μ	chemical potential, p. 20
ν	kinematic viscosity, p. 26
ρ	(mass) density, $\rho = 1/\alpha$, p. 12
$\sigma_{\Delta t}$	physical time step CFL number, p. 83
$\sigma_{\Delta\tau}$	pseudo-time step CFL number, p. 84
τ	pseudo-time, p. 81
τ_V	stabilization matrix for V variables, p. 52
$\psi^{n,g}$	continuous spatial global basis function at node g in time slab n, p. 48
ψ	intensive quantity: density of Ψ, p. 12
$\psi^n_{(e,\tilde{k},f)}$	f^{th} basis function for the $\tilde{k}^{\text{th}}$ solution component on the element $\mathcal{K}_e^n$
$\hat{\psi}$	Chapter 2: specific value of Ψ (intensive quantity), Chapter 5: basis function on the reference element

Symbols starting with an uppercase Roman letter

A_i^V	Jacobian with respect to V-variables of the Euler flux F_i^{e} in the i^{th} coordinate direction, p. 27
B_{bc}	boundary term in the least-squares FEM, p. 48
B_{ls}	least-squares term, p. 48
$\mathbb{C}$	set of complex numbers
C_{IP}	stabilization parameter of the IP method, p. 76
$\mathcal{D}$	stability domain (of a RK method), p. 83
$\mathbb{D}$	rate of deformation tensor or rate of strain tensor, p. 15
D_{r}	diagonal matrix with reference velocity powers, used in the entropy variable scaling, p. 35
E	internal energy, p. 12
Ec	Eckert number, (2.50), p. 31
$\mathcal{E}$	space-time domain, p. 39
$\mathcal{E}^n$	space-time slab (also just called *time slab*), p. 40
$F_{e,j}^n$	space-time element face of the element $\mathcal{K}_e^n$, p. 43
$F_{\bar{\imath}}^{\mathsf{d}}$	diffusive flux in the $x_{\bar{\imath}}$-coordinate direction, p. 27
$F_{\bar{\imath}}^{\mathsf{e}}$	Euler flux in the $x_{\bar{\imath}}$-coordinate direction, p. 27
$\hat{F}$	numerical flux function of the flux F, p. 69
$\boldsymbol{F}$	external body force, p. 14
$\boldsymbol{F}_\psi$	non-convective flux density of ψ, areal molecular flux, p. 13
$\bar{F}_{e,j}^{n-1,+}$	space element face of the element $\bar{\mathcal{K}}_e^{n-1,+}$, p. 42
G_A^B	mapping from A to B, p. 42
$\bar{G}_e^n$	time-dependent mapping from the space reference element $\hat{\bar{\mathcal{K}}}_e^n$ to the space element $\bar{\mathcal{K}}_e^n(t)$, p. 42
$\bar{G}_e^{n-1,+}$	mapping from the spatial reference element $\hat{\bar{\mathcal{K}}}_e^{n-1}$ to the physical space element $\bar{\mathcal{K}}_e^{n-1,+}$, p. 40
G_e^n	mapping from the space-time reference element $\hat{\mathcal{K}}_e^n$ to the corresponding element $\mathcal{K}_e^n$, p. 42
$G_{\hat{\mathsf{T}}}^{\mathsf{T}^n}$	mapping from reference to physical time for space-time slab n, p. 42
$\mathcal{H}^n$	hyperplane, p. 40
$H^k(\mathcal{E}_h^n)$	Sobolev space on $\mathcal{E}_h^n$, cf., e.g., (Brenner and Scott, 2002)
$I_{d\times d}$	$d \times d$ identity matrix
$\Im(\mu)$	imaginary part of the complex number μ
$\mathbb{I}$	unit tensor
$K_{\bar{\imath}\bar{\jmath}}^V$	Jacobian of the diffusive flux in the i^{th} coordinate direction, F_i^{d}, with respect to the spatial derivative $\partial V/\partial x_{\bar{\jmath}}$ of the V variable set, p. 27

$\bar{\mathcal{K}}_e^{n-1,\pm}$	e^{th} space element in $\bar{\mathcal{T}}^{n-1,\pm}$, p. 40
$\hat{\bar{\mathcal{K}}}_e^{n-1}$	reference element of $\bar{\mathcal{K}}_e^{n-1,+}$ (and $\bar{\mathcal{K}}_e^{n,-}$), p. 40
$\mathcal{L}^{\mathrm{e},n}, \mathcal{L}^{\mathrm{d},n}, \mathcal{L}^n$	inviscid (Euler)/viscous/Navier–Stokes operator in space-time slab n, p. 82
$\mathcal{L}$	fundamental quantity length, p. 30
$L^2(\mathcal{E}_h^n)$	space of square Lebesgue integrable functions on $\mathcal{E}_h^n$, p. 68
$\mathcal{M}$	fundamental quantity mass, p. 30
M	Chapter 2: mass of a system, otherwise: Mach number
N_{el}^n	number of elements in the n^{th} space-time slab, p. 40
N_t	number of time subintervals, p. 40
$P_{(p_{\mathrm{t}},p_{\mathrm{s}})}(\hat{\mathcal{K}}_e^n)$	space of tensor product polynomials of order p_{s} in space, augmented with monomial basis functions in time up to order p_{t} on the reference element $\hat{\mathcal{K}}_e^n$, p. 47
$\mathbb{P}$	stress tensor (also pressure tensor), p. 15
Pr	Prandtl number, (2.49), p. 30
$\mathcal{P}_h^{n,(\langle p_{\mathrm{t}}^{n,e}\rangle_e,\langle p_{\mathrm{s}}^{n,e}\rangle_e)}$	function space of the DG method on the n^{th} discrete space-time slab with temporal and spatial orders $p_{\mathrm{t}}^{n,e}$ and $p_{\mathrm{s}}^{n,e}$ on the elements $\mathcal{K}_e^n$, p. 68
$\mathbb{P}_{\mathrm{visc}}$	viscous stress tensor, p. 15
$\mathcal{Q}$	hypersurface spanned by the spatial domain boundary $\partial\Omega(t)$ and time, p. 39
$\mathcal{Q}_h$	space-time boundary of the triangulated domain, p. 41
$\Re(\mu)$	real part of the complex number μ
$\mathbb{R}$	set of real numbers
R	gas constant, p. 22
Re_η	Reynolds number, (2.48), p. 30
Re_λ	dimensionless number involving the dilatational viscosity, (2.47), p. 30
$\mathrm{Re}_{\mathrm{el}}$	element Reynolds number, p. 58
S	source term (in the Euler and Navier–Stokes equations), p. 27
$\mathcal{S}_{e,j}^{\mathrm{b},n}$	boundary face, p. 43
S_{e}	external volumetric rate of supply density, p. 13
$\mathcal{S}_{e,e'}^{\mathrm{f},n}$	future time face, p. 43
$\mathcal{S}_{\{e,e'\}}^{\mathrm{i},n}$	internal space-time face, p. 43
S_{i}	internal volumetric production rate density, p. 13
$\mathcal{S}_{e,e'}^{\mathrm{p},n}$	past time face, p. 43
T	time interval, p. 40
T	temperature, p. 12
T_{c}	critical temperature of a vdW gas, p. 24
$\hat{\mathsf{T}}$	reference time interval, p. 42
T^n	time subinterval, p. 40

$\mathcal{T}$	fundamental quantity time, p. 30
$\bar{\mathcal{T}}^{n,\pm}$	space tessellation on the ± side of $\mathcal{H}^n$, p. 40
U	conservation variables, (2.44), p. 28
$\mathcal{V}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}$	trial/test function space of the DG method for the flux form equation, p. 68
V	Chapter 2: volume of a system, otherwise: entropy variables (also: generic variable set)
V^b	boundary data, p. 76
$\check{V}^n$	all expansion coefficients of the discrete representation of V in the n^th space-time slab, p. 80
$\check{V}^n_e$	expansion coefficients of the discrete representation of V on $\mathcal{K}^n_e$, p. 80
$V_g^{n,j}$	expansion coefficient (vector) with respect to the global spatial basis function $\psi^{n,g}$ and time basis function $\lambda^{n,j}$, p. 49
$V_{i,e}^{n,j}$	element local expansion coefficient (vector), p. 49
$V(t)$	(moving) material volume at time t, p. 12
$\mathcal{W}$	fundamental quantity temperature, p. 30
$\mathcal{W}_h^{n,(p_\mathrm{t},p_\mathrm{s})}$	trial function space of the Galerkin least-squares method, p. 47
W	test function, p. 48
$\mathcal{X}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}$	trial/test function space for the auxiliary variable of the DG method, p. 68
$\mathcal{Y}_h^{n,(p_\mathrm{t},p_\mathrm{s})}$	test function space of the Galerkin least-squares method, p. 47
Y	primitive variables with pressure, (2.46), p. 28
Y'	primitive variables with density, (2.45), p. 28

Symbols starting with a lowercase Roman letter

a	sound speed, also: parameter in the vdW EOS, p. 23
b	parameter in the vdW EOS, p. 23
c_p	specific heat at constant pressure, p. 20
c_v	specific heat at constant volume, p. 20
d	space dimension of the considered problem, p. 18
đ	inexact differential, p. 20
e	specific internal energy, p. 14
e^tot	total energy, sum of internal and kinetic energy, p. 19
$\boldsymbol{e}_{\bar{\imath}}$	unit vector in the $\bar{\imath}^\mathrm{th}$ Cartesian coordinate direction
$\boldsymbol{g}$	acceleration of gravity, p. 14
g	global nodal index, p. 48
h	Chapter 2: specific enthalpy, Section 5.2.9: time step of a RK method
i	index for space-time dimensions, $i \in \{0, \ldots, d\}$, p. 18
$\bar{\imath}$	index for spatial dimensions, $\bar{\imath} \in \{1, \ldots, d\}$, p. 18

k	kinetic energy density, p. 14
$\boldsymbol{n}$	outward unit normal vector
n_t	number of temporal basis functions in a slab, p. 49
p	(thermodynamic) pressure, p. 12
p_c	critical pressure of a vdW gas, p. 24
p_s	spatial order of an expansion
p_t	temporal order of an expansion
$\boldsymbol{q}$	heat flux density vector, p. 15
$\boldsymbol{r}$	position vector, p. 12
s	specific entropy, p. 20
t	time
t_e	end time, p. 39
t_s	start time, p. 39
$\boldsymbol{v}$	velocity vector
x	point or coordinate vector, p. 39
x_0	temporal coordinate of point $\boldsymbol{x}$, p. 18
$x_{\bar{\imath}}$	spatial coordinate $\bar{\imath}$ of $\boldsymbol{x}$ with respect to the Cartesian coordinate system, p. 18
$\hat{x}$	reference space-time point, p. 42
$\hat{x}_0$	reference time, p. 42
$\hat{\hat{x}}$	reference space point, p. 42
$\boldsymbol{x}$	position vector, p. 18

Other symbols

$\{\!\{\cdot\}\!\}$	average operator, p. 68
$\partial V(t)$	boundary of the volume $V(t)$, p. 13
$(\cdot)_\infty$	far-field value
$[\![\cdot]\!]$	jump operator, p. 69
$(\cdot)^\mathsf{L}$, $(\cdot)^\mathsf{R}$	data from the left/right side of a face
∇	nabla operator
$(\cdot)_\mathsf{r}$, $(\cdot)_{\mathsf{r}'}$	reference value, p. 30
$\langle p_\mathsf{t}^{n,e} \rangle_e$	sequence of the temporal expansion orders on all elements in the n^{th} space-time slab, p. 68
$(\cdot)^*$, $(\cdot)'$	dimensionless number, p. 30
$(\cdot)^\pm_{\mathcal{K}_e^n}$	internal/external trace with respect to the element $\mathcal{K}_e^n$, p. 43
$(\cdot)^\mathsf{T}$	transpose of a tensor or matrix

Chapter 1

Introduction

Do not imagine that mathematics is hard and crabbed,
and repulsive to common sense.
It is merely the etherealization of common sense.

William Thomson (1824–1907)

1.1 Motivation

The flow of fluids is a ubiquitous phenomenon in our environment. Not only is life as we know it dependent on fluids in countless ways, also human inventions exploit the physical and chemical properties of gases and liquids in innumerably many ways. The urge to understand the relevant processes has born the field of fluid dynamics. While many single fluid flows are well comprehended nowadays, the details of flow with more than one fluid involved are naturally more complex and still defy a thorough understanding and control. A particular class of such flows are those with free interfaces, i.e., two or more fluids that do not mix, but maintain a relatively sharp interface between each other. An everyday life example are flows with bubbles. The interest in such flows derives from the fact that the characteristics of the flow of the composite can be strikingly different from a single component case. The understanding of how such processes depend on the multifluid aspects is crucial to the exploitation of such effects.

Just as in other branches of physics, research in fluid dynamics can be characterized by a tripolar structure of theory, experiment, and—since the availability of computers—numerical simulation as modi operandi. Already for single fluid flow, theoretical approaches typically allow only statements about idealized configurations, and it will be even more difficult to apply them to realistic multiphase flow situations. Experiments, on the other hand, are often hard and expensive to realize, with complications in acquiring measurements of the desired quantities and the reproducibility of results. Therefore, numerical simulation plays an important role in the understanding of multiphase flow.[1]

[1] The term *multiphase flow*, which literally refers to a special class of multicomponent flows, is used interchangeably with multicomponent or multifluid flow here. The challenges considered in this chapter apply to all flows of several immiscible fluids.

Simulation of multiphase flow is carried out in different ways, depending on the scale of the examined effects. The available computer capacity limits the domain size and the amount of detail that can be captured. For industrial-size configurations, disperse gas-liquid (bubbly) flow is usually modeled in a purely Eulerian manner by deriving averaged equations for the complete fluid mass, see, for example, (Drew and Passman, 1998). The averaging operation for the nonlinear terms in the continuum equations leads to undetermined correlation terms, which have to be modeled by closure relations, i.e., parameterizations in terms of known characteristics of the flow. The parameterizations have to be calibrated with the help of theory, measurements, or simulations with a resolution that allows to explicitly capture the relevant processes. Given the same computing limitations, simulations of the latter type use a much higher resolution than those based on models with parameterizations and consequently they will allow only a limited number of bubbles or other structures to be included. On the other hand, at this scale the number of properties and processes that is included explicitly in the model can be much larger. Features that could be taken into account (without parameterization) in micro-scale models but not in macro-scale ones are,[2] for example, the physics and chemistry of the individual fluids, changes in bubble shape, bubble interactions with or without changes in topology (splitting and merging), and (chemical) processes at the interface. Apart from the computer bandwidth lagging behind the demand, also the available mathematical descriptions limit the simulations: the model—and ultimately the numerical method used to evaluate it—must be able to deal with the effects one wishes to consider.

In the described problem domain, the current work focuses on the development of a single-fluid numerical method for the Navier–Stokes equations that particularly addresses some of the abovementioned issues of the multifluid context. A few items that will receive attention are singled out next.

First of all, regarding the physics of individual fluids, many studies assume incompressibility for both phases in gas-liquid flow, even under conditions where one would expect at least the gas phase to be compressible. The simplification is a consequence of the dichotomy of computational fluid dynamics, with methods for compressible flow on the one hand and those for incompressible flow on the other. Hauke and Hughes (1998) describe the separation of the two fields by almost every part of the numerical algorithms: Starting from the formulation of the flow equations, over the discretization method to the solvers for the finite-dimensional equation system the components of compressible solvers typically differ from those of incompressible ones.

As a consequence of the disparity between numerical methods for compressible and incompressible flow, for the application to multiphase flows there are several options: First, one might couple methods specialized for the different phases at the fluid interfaces.

[2]The list of processes is based on the available computing power at the time of writing. In the future, larger-scale models may be able to include such effects without parameterizations.

Alternatively, a different and more general kind of method than typically applied may be sought, namely one that suits both cases. A third way is to drop the continuum model (here the Navier–Stokes equations) and switch to a more fundamental physical viewpoint, as is done in Lattice-Boltzmann methods, see, e.g., (Chen and Doolen, 1998). The last option is discarded as a matter of choice, the goal of this thesis is to develop a numerical method based on the fluid description with the Navier–Stokes equations. The first option, the coupling of numerical methods for different fluids at interfaces, is expected to be difficult, both theoretically (e.g., regarding the consistency at the interface) and concerning its implementation. Hence, the second way is chosen, i.e., a formulation suitable for different types of fluid is sought. The first central question therefore is:

Central question 1:
How to obtain a simulation tool that is applicable for a wide range of physical conditions and for different fluids, especially for both compressible and incompressible flow?

Given a mathematical model formulation that can cope with different physical conditions, the next challenge in the description of fluid flow is its multi-scale nature. Already in single-phase flow, and even more so in multiphase configurations, a wide range of scales is involved. Theoretically one ought to include features of the scale of the outer domain boundary dimension, over the possible bubble and interface sizes down to the dissipative scale of the (turbulent) flow. Distinct flow features like vortices may emerge, evolve, and decay in time. The numerical method used to solve the mathematical model of the flow has to be able to capture these phenomena and to cope with their movement and transience. In the context of physical space discretizations, this task calls for dynamic grid adaptation. When considering multiphase flow, the interfaces between portions of the different fluids constitute an additional complication. Several methodologies exist to capture these interfaces. To obtain an accurate description of the interface movement at reasonable computational cost, the geometric mesh that is used for the computations has to be locally refined in the proximity of the interface. For the described problems, the numerical method has to allow adapting the resolution locally and temporarily. Therefore, a second point of consideration is formulated as follows:

Central question 2:
Given a mathematical model for different fluids, which numerical method possesses the geometric flexibility to lend itself well to the simulation of multi-scale effects and interfacial features?

Apart from the previous two questions, which are mainly inspired by the processes whose simulation is the goal, a third issue is added. The topic of this question is related to the translation of the numerical method into a computer program, which is a requirement for using the algorithm for simulations. The implementation step typically takes up significant resources. Developing, testing, and documenting code is a time-consuming undertaking, and ever more so as the complexity of algorithms and computer architectures is increasing.

For these reasons, a part of this work addresses a third question, which is posed very generally as:

Central question 3:
How to facilitate the implementation step of the developed numerical methods?

The findings regarding this question benefit both the implementation of the mathematical model developed for the flow problems described earlier as well as a much wider class of numerical methods outside the scope of this thesis.

The work presented in this dissertation is evaluated by its meaningfulness regarding the three central questions, which will resurface repeatedly. Ultimately, an answer emerges for each question. In the remainder of this introductory chapter, the individual topics are explained in detail and related to previous research. Each central issue receives some more attention, and the problem as well as the approach taken for its solution is described.

1.2 A continuum description for the numerical simulation of compressible and incompressible fluids

The physical model adopted here is given by the Navier–Stokes equations. This set of time-dependent nonlinear second order partial differential equations is sufficiently general for the purposes pursued: The equations constitute the mathematical formulation of the physical conservation statements for mass, momentum, and energy, which apply to all fluids. Further, the system includes the restriction to Newtonian fluids and transport, viscous, and Fourier heat conduction phenomena only. The Navier–Stokes equations are not closed, though: they contain more variables than equations. For the closure of the system, equations of state are used. These relations describe the thermodynamical state of a fluid. Different media have different equations of state, from which again different problems may arise for a numerical method. Many numerical methods for fluid dynamics are tailored to overcome one particular problem, with the result that they are suitable only for a single, idealized type of fluid. Most prominently, algorithms for compressible flow are frequently tailored to ideal gases, and another class of schemes is designed for incompressible media. Both underlying fluid models are important as they capture well the essential behavior of many fluid flows. At the same time the restriction to one or the other case means that the numerical method cannot be applied for other media (e.g., real gases) or for situations in which several fluids with different properties populate the domain in the form of a fluid composite (e.g., bubbly flow). Furthermore, many flow schemes originally designed for compressible flows suffer from convergence and accuracy problems in the low Mach number (M) limit (Guillard and Viozat, 1999; Guillard and Murrone, 2004), which hampers their application to cases in which high and low Mach numbers coexist. The goal here is to develop a numerical method that can be applied to different media (i.e., different equations of state) and various physical conditions, including high as well as low Mach number flows.

Returning to the low Mach number problem, several solutions have been devised. One approach modifies the equations by using a series expansion in the Mach number; like any finite series, however, the resulting description is limited to a neighborhood of the expansion point $M = 0$, so that applicability under some circumstances is achieved at the cost of other conditions, namely the higher subsonic and supersonic Mach number range. Another approach starts from the compressible Navier–Stokes equations and aims at making a numerical method constructed for these equations suitable for the incompressible limit. This can be accomplished, e.g., by preconditioning or by the introduction of artificial compressibility. Preconditioning, see (Turkel et al., 1997; Guillard and Viozat, 1999; Turkel, 1999; Guillard and Murrone, 2004) and references therein, aims at reducing the disparity between the convective and sound wave speeds and the different scaling of the pressure and velocity field as a function of the Mach number. However, when taking this technique to high Mach numbers, problems with a singular preconditioning matrix occur at stagnation points and sonic lines. Furthermore, preconditioning matrices are cumbersome to derive, depend on the numerical method used, and lead to schemes that may be thermodynamically inconsistent (Hauke and Hughes, 1998). Artificial compressibility, on the other hand, introduces a parameter into the continuity equation that mimics physical compressibility by coupling pressure with density. Unfortunately, the resulting numerical methods have a wrong time dependence, unless the artificial compressibility is introduced in the context of a pseudo-time stepping method (Soh and Goodrich, 1988). Finally, a third type of schemes for low Mach number problems starts from a solver for the incompressible Navier–Stokes equations and adapts it to the compressible case while making sure that the well-defined limit is preserved, see, e.g., Bijl and Wesseling (1998) (and references therein), who rewrite the Navier–Stokes equations in terms of the independent variables pressure, momentum, and enthalpy, and apply a pressure correction method.

In the present work, an ansatz is followed that combines some of the characteristics of the mentioned approaches but takes a different starting point. The compressible Navier–Stokes equations are used, noticing the approach to conservation laws introduced by Godunov (1962): By performing a change of variables, the system matrix of a (quasi-) linear system of partial differential equations is symmetrized if the special set of *entropy variables* is used. The symmetrized form of the governing equations has several mathematically and physically interesting properties, cf. (Hauke and Hughes, 1998), in the case of fluid mechanics especially that it yields a formulation suitable for a wide range of physical conditions. Aside from the broad applicability of the numerical method, another advantage is the automatic satisfaction of the second law of thermodynamics (Hughes et al., 1986). These topics are addressed in Chapter 2. For background on the symmetrization of hyperbolic systems and the derivation of the entropy variables the work of Barth (1999) is recommended and provides further references.

1.3 Geometrically flexible discretization with finite element methods

One of the topics raised in Section 1.1 is the presence of widely different scales in (multiphase) flows. On the one hand this concerns the flow features (vortices, boundary layers, etc.) in single fluid problems, and on the other hand the fluid-fluid boundaries in the multifluid context.

Numerical methods solve the mathematical equations on a discrete representation of the flow domain, a *mesh*. In combination with the previously mentioned challenges, the mesh has to offer more degrees of freedom and higher resolution in the area of the fine scale features of the flow than in the parts of the domain where the flow is largely homogeneous. Apart from the functionality to allow such adaptations, tracking an embedded interface has to be feasible for multifluid cases. These requirements cannot be realized with standard equal-spaced meshes. A promising approach to local refinement is provided naturally by *unstructured meshes*. In such meshes, the individual cells may vary in geometry, basic shape, and connectivity in the mesh. Consequently there is no general rule to find neighboring cells as in structured meshes and the information about neighbors has to be stored for each cell.

The varying cell geometry in an unstructured mesh has to be supported by the numerical method that is supposed to work on it. A family of techniques that is particularly suited for this purpose are finite element methods. These have the additional benefit that they can be applied to hyperbolic, parabolic, and elliptic partial differential equations, all three types having their own mathematical peculiarities. Consequently, finite element methods have been used frequently for the discretization of the Navier–Stokes equations, which are of incomplete parabolic type. For nonlinear hyperbolic partial differential equations (PDEs), the classical Galerkin finite element method (FEM) lacks stability. This deficiency can be amended in several ways. One of them is the addition of a least-squares term, and this route is followed by parts of the current work. More recently, however, these methods have faced a competitor in the form of *discontinuous Galerkin methods*. These allow more general solution spaces and are even better suited for unstructured meshes, as they eliminate the continuity requirement of classical Galerkin FEMs at the inter-element boundaries.

The possibility of choosing discontinuous (basis) functions is exploited, in the first place, when using local refinement with so-called *hanging nodes*, see, for example, (van der Vegt and van der Ven, 2002b). Hanging nodes pose additional problems in continuous methods and thus complicate the implementation of refinement and of meshes with cells of different shape. Without doubt, the possibility of refinement and mixed meshes will be of great use in the context of tracking small scale structures and embedded interfaces. For this purpose, the current work should be linked to the research of Sollie et al. (2007), who develop a method to track interfaces in space and time with a discontinuous Galerkin method. In this dissertation, the focus is laid on the combination of a discretization for the Navier–Stokes equations with the solver for the nonlinear algebraic system of equations. A pseudo-time integration is used for this purpose. This choice is in line with earlier research in space-time discontinuous Galerkin finite element methods (van der Vegt and van der Ven, 2002b; van der Ven and van der Vegt, 2002; Klaij et al., 2006a), but different from the typical Newton (-like) procedure often used in classical Galerkin methods, e.g. (Hauke and Hughes, 1998; Barth, 1999). The reasons for this choice and the adaptations necessary for the entropy variable approach are the subject of Chapter 5.

1.4 Software tools for the implementation of discontinuous Galerkin finite element methods

The design of a finite element (FE) algorithm starts with the formal definition of the method by deriving a weak formulation of the system of partial differential equations and choosing basis functions to discretize the function spaces, cf. Chapters 4 and 5. Properties of the FEM can then be examined and, given satisfying results, the next step is to use the method for solving the target problem numerically. For that purpose, the developed algorithm has to be translated into a computer program. A correct implementation not only provides a numerical solution but can also be used to determine additional properties of the algorithm, like approximation orders, iterative convergence rates, and computational costs.

The definition and analysis of a FEM is a complicated exercise and relies on the mathematical skills of the developer. The third central question (cf. p. 4) addresses the following step: the implementation as a computer program, which can be considered an equally complex task. Additional complications arise from developments on the computer hardware side: increasingly powerful computing systems on the one hand allow the numerical solution of many real-world problems; on the other hand, the utilization of such systems poses additional requirements on the implementation—parallelization is a keyword that alludes to the added complexity. At this point it is tenable that the transformation of a mathematical model into a capable computer code is a task that goes far beyond the abilities of a single, say mathematically trained, person. Apart from the skill constraint, developments are also limited by the amount of work that an individual can accomplish on the timescale of a typical research project period. The logical consequence is that efforts have to be joined to reach the forefront of current developments in applied mathematics.

A related difficulty regarding the efficiency of the software development process is to maintain productivity over more than a single project. Having invested in software design and development as described above, the effort would be wasted if there was no possibility to reuse the result, i.e., the software artifacts, for applying new algorithms to the same or other problems. Therefore it is important that software is built in a modular and extensible way, representing general concepts and separating the application-related items from abstract mathematical parts, and those in turn from underlying computer-scientific details. Hence it is a prerequisite to decompose FEMs into recurring components and tasks, and additional parts, which are specific to individual methods.

The foundation of hpGEM, the software framework presented in Chapter 6, is that those parts that are the same in many FEM implementations constitute a relatively large fraction. Examples are the representation of the geometric mesh of the domain, the mathematical definition of the finite elements (shape, basis functions, and degrees of freedom, see, e.g., (Ciarlet, 1978; Brenner and Scott, 2002)), and the assembly and solution of systems of equations. The crux is that typically these parts are 'reinvented' and reimplemented by individuals when starting from scratch, incurring a large overhead in development time and—more importantly—the potential of introducing coding errors. The hpGEM framework, by contrast, provides well-tested data structures and methods on which the specific FE application can build, thus cutting short the implementation time and reducing the danger of introducing mistakes.

Naturally, the decomposition of FEMs also reveals tasks for which highly developed, tested, and established solutions are available, e.g. from computer science (data containers and search algorithms) or numerical linear algebra (solvers for systems of equations). It is not so much the question *whether* to use them but rather *how* to do it. Here the benefit of the presented software environment is that it provides access to various packages, which enhance its functionality and capabilities.

1.5 Overview of this thesis

In the first section of this chapter, three central questions were formulated which set out the lines that the investigations follow. In Chapter 2, the model for the considered physical systems is formulated and the extension to generalized variable sets, which enable to construct numerical methods for both compressible and incompressible flows, is described.

Common notation and definitions are established in Chapter 3 before two numerical methods are presented. The first one builds on earlier research by Polner (2005) and Polner et al. (2006) on a Galerkin least-squares finite element method for the generalized variable approach. Their research focused on the definition of a stabilization operator that matches the entropy variable formulation in its applicability to compressible and incompressible fluids. This goal was reached by Polner, Pesch, and van der Vegt (2007), to which findings from Chapter 4 of the current thesis contributed. At that point—although the performance of the method was satisfactory for the simulation of different single fluids, doubt arose whether the method would be able to satisfy the demands set out for the current work. The most important reason for these doubts is the limited flexibility in conjunction with locally refined meshes.

Therefore the focus of the current project shifted away from the Galerkin least-squares finite element method and turned towards a discontinuous Galerkin discretization. The latter type of finite element method is applied to the chosen model in Chapter 5. Starting from the space-time discontinuous Galerkin method for the Euler equations by van der Vegt and van der Ven (2002b), some modifications were necessary in connection with the generalized variable approach and different equations of state as pointed out by Pesch and van der Vegt (2007). In addition, the current thesis contains the extension to the Navier–Stokes equations and several numerical examples for different fluids.

Chapter 6, introduces hpGEM, an object-oriented software framework for discontinuous Galerkin finite element methods. Material presented by Pesch et al. (2007) is reconsidered and extended, describing the general ideas and structure. A few solutions are treated in more detail and examples are given how an application code—in this case the Navier–Stokes solver from Chapter 5—can be built upon the framework.

Conclusions are drawn in Chapter 7 regarding the three central questions posed for this thesis. Based on the results of this dissertation, directions and objectives for further research are suggested.

Chapter 2

Thermodynamical and fluid dynamical foundations

Niets is volmaakt, niets is ten einde,
elke verstarring is bedrog:
alles wat was, vloeide in het zijnde,
alles wat is, vloeit nog.

Herwig Hensen (1917–1989)

2.1 Introduction

The modeling of fluids can proceed in many ways and different applications require qualitatively different statements. In one context, the movement of fluids may be of interest, for example for transport phenomena or to evaluate the dynamic interaction of the flow with fixed or moving structures. In other areas, the kinetics may be negligible, but the fluid interaction with its environments is of interest, e.g. through exchange of heat or work.

For studying different processes, it would be desirable to possess a model that is able to capture as many of these phenomena as possible. To conform to this target, a general description has to be used, within the bounds set out by physical understanding and mathematical feasibility. In Section 2.2, the fundamental physical rules of conservation of quantities like mass, momentum, and energy are applied to test volumes of a fluid, which results in a space- and time-dependent description. The resulting mathematical equations contain several unknown quantities; some of these—the non-convective flux terms—are modeled by standard assumptions in Section 2.3. The system of equations is stated in vector form and Cartesian component form in Sections 2.4 and 2.5, respectively. Furthermore, also the thermodynamical state of the fluid has to be described, which is subject of Sections 2.6 and 2.7.

An extension of the conservation law treatment is taken up in Section 2.8: The independent variables in the equations are not an intrinsic property of the equations, and using different sets of variables can lead to formulations with special properties. In particular, some choices of variable set allow a wide range of applicability, e.g. for both compressible and incompressible fluids.

A discussion of the dimensional aspects of the conservation equations follows in Section 2.9. The physical equations are valid independent of the set of measuring units, and by non-dimensionalizing the equations the choice of units can be subsumed in a number of dimensionless quantities that are introduced in Section 2.10. Conclusions for this chapter and an outlook on its implications for the following chapters are given in Section 2.11.

2.2 Conservation laws

To obtain a mathematical description of fluid flow, physical conservation laws are applied to evaluate the balance of quantities like mass, momentum and energy. For example, for momentum, Newton's law states that the temporal change of the momentum of a body is equal to the sum of the forces on it. Such a balance may be applied to the overall system, but the goal here is to model the behavior of a fluid in space and time. Hence, the system is conceptionally divided into smaller subsystems whose interaction is quantified in the balance equations. The subsystems may be moving: a *material volume* $V(t)$ is transported (or 'convected') with the local fluid velocity so that at all times the deforming volume as well as its surface consists of the same fluid particles. Quantities that depend on the mass contained in a subsystem are called *extensive* and denoted by uppercase letters; examples are the volume V occupied by the system, the internal energy E, and the system mass M. On the other hand, *intensive* quantities are those that do not depend on the system mass. Intrinsically intensive quantities are, e.g., *pressure* p and *temperature* T. Extensive quantities have intensive counterparts, which are obtained by dividing the extensive quantity by the system mass or volume. Intensive quantities are denoted by lowercase letters.[1] An intensive quantity that results from dividing an extensive quantity Ψ by the system mass M is called *specific value of* Ψ and denoted with the corresponding lowercase letter $\psi = \Psi/M$. For example, the *specific volume* is defined as $\alpha = V/M$. If an intensive quantity is obtained by division by the system volume V it is called the *density of* Ψ and denoted[2] $\hat{\psi} = \Psi/V$, e.g. *(mass) density* $\rho = M/V = 1/\alpha$. With these notational conventions, the amount Ψ of some extensive physical property of arbitrary tensorial rank contained in the material volume $V(t)$ at time t is

$$\Psi(t) = \int\limits_{V(t)} \hat{\psi}(\boldsymbol{r}, t)\,\mathrm{d}V = \int\limits_{V(t)} \rho\psi\,\mathrm{d}V\,. \tag{2.1}$$

The rate of change of the integral property is expressed as

$$\frac{\mathrm{d}\Psi}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{V(t)} \rho\psi\,\mathrm{d}V = \int\limits_{V(t)} \frac{\mathrm{D}}{\mathrm{D}t}(\rho\psi)\,\mathrm{d}V = \int\limits_{V(t)} \left(\frac{\partial\rho\psi}{\partial t} + \nabla\cdot(\rho\psi\boldsymbol{v})\right)\mathrm{d}V\,, \tag{2.2}$$

[1] Temperature T is the main exception to the rule; others may occur in later chapters when the focus is not on the physical modeling but on the mathematical methods.

[2] In the sequel it will not always be textually specified whether a statement applies to a density or specific value of a quantity, as the distinction is normally clear from the context or notation.

property Ψ	density of Ψ $\hat{\psi}$	non-convective areal flux $\boldsymbol{F}_\psi$	internal production rate S_i	external supply rate S_e
mass	ρ	0	0	0
linear momentum	$\rho\boldsymbol{v}$	$-\mathbb{P}$	0	$\boldsymbol{F}$
total energy	$\hat{e} + \frac{1}{2}\rho\boldsymbol{v}\cdot\boldsymbol{v}$	$\boldsymbol{q}$	$-\nabla\cdot(\boldsymbol{v}\cdot\mathbb{P})$	$\boldsymbol{F}\cdot\boldsymbol{v}$

Table 2.1: Expressions for the terms in the balance equation (2.4) for several properties. The involved quantities are: mass density ρ, velocity $\boldsymbol{v}$, stress tensor $\mathbb{P}$ (cf. Section 2.3.1), external body force $\boldsymbol{F}$ (e.g. $\boldsymbol{F} = \rho\boldsymbol{f} = \rho\boldsymbol{g}$ in case of gravity), e specific internal energy, and the heat flux vector $\boldsymbol{q}$ (cf. Section 2.3.2). Adapted from (Edwards et al., 1991).

where the *budget operator* D/Dt and the velocity $\boldsymbol{v}$ are introduced. The budget operator takes into account the local changes at fixed positions as well as the changes due to the deformation of the surface of the volume, see, e.g., (Zdunkowski and Bott, 2004).

The material derivative $\mathrm{d}\Psi/\mathrm{d}t$ can be split up into two components, representing the changes due to external and internal interactions or processes, $\mathrm{d_e}\Psi$ and $\mathrm{d_i}\Psi$, respectively,

$$\frac{\mathrm{d}\Psi}{\mathrm{d}t} = \overbrace{\frac{\mathrm{d_e}\Psi}{\mathrm{d}t}}^{} + \overbrace{\frac{\mathrm{d_i}\Psi}{\mathrm{d}t}}^{}$$

$$= \underbrace{-\oint_{\partial V(t)} \boldsymbol{F}_\psi \cdot \boldsymbol{n}\,\mathrm{d}(\partial V) + \int_{V(t)} S_\mathrm{e}\,\mathrm{d}V}_{\frac{\mathrm{d_e}\Psi}{\mathrm{d}t}} + \underbrace{\int_{V(t)} S_\mathrm{i}\,\mathrm{d}V}_{\frac{\mathrm{d_i}\Psi}{\mathrm{d}t}}, \tag{2.3}$$

with the non-convective flux density $\boldsymbol{F}_\psi$ (areal molecular flux), the (external) volumetric rate of supply density S_e, the (internal) volumetric production rate density field S_i, and the outward unit normal vector $\boldsymbol{n}$. By applying the divergence theorem to the surface integral in (2.3), the balance equation (2.2) becomes

$$\int_{V(t)} \left[\frac{\partial\rho\psi}{\partial t} + \nabla\cdot(\rho\psi\boldsymbol{v} + \boldsymbol{F}_\psi) - S_\mathrm{i} - S_\mathrm{e}\right] \mathrm{d}V = 0\,. \tag{2.4}$$

Arguing that the integration volume V in (2.1) may be chosen arbitrarily within the fluid, the integrand of (2.4) is required to vanish at all points in space and time, thus leading to a partial differential equation for the evolution of the intensive quantity ψ:

$$\frac{\partial\rho\psi}{\partial t} + \nabla\cdot(\rho\psi\boldsymbol{v} + \boldsymbol{F}_\psi) - S_\mathrm{i} - S_\mathrm{e} = 0\,. \tag{2.5}$$

The entities occurring in Eq. (2.5) are listed in Table 2.1 for the properties considered later in the Euler and Navier–Stokes equations. The physical considerations leading to their specification are summarized in the following paragraphs.

Remark 2.1 *The equations can also be derived for a fixed volume V_0 with surface S_0, but the resulting* differential *equation, unlike the integral one, is the same.* □

Remark 2.2 *The classical notion of differentiability of the field variables may not allow the transition from the integral to differential equations, namely when the sought functions contain discontinuities.* □

Mass

When the fluid is considered as a single bulk substance, there is no net non-convective flux. Mass can neither be created nor destroyed by internal or external influence, which is reflected by the *continuity equation*,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0\,. \tag{2.6}$$

Momentum

Due to the interaction of the molecules of a fluid its motion evokes forces, which may redistribute *momentum* $\hat{\psi} = \rho \boldsymbol{v}$ in the fluid. Consequently, through every (imaginary) surface in the fluid, there is a momentum flux $\boldsymbol{F}_\psi$. By physical considerations one deduces that the *surface force* per unit area, which is frequently called *stress* in the physics of fluids and solids, in each point of the surface can be represented as the scalar product $\mathbb{P} \cdot \boldsymbol{n}$ of a symmetric rank two tensor $\mathbb{P}$ and the local surface normal $\boldsymbol{n}$, see, e.g., (Warsi, 1999). This product can be readily incorporated into Eq. (2.3). The properties and functional dependence of the tensor $\mathbb{P}$ are discussed further in Section 2.3.1.

Additionally, the flow may be modified by external body forces $\boldsymbol{F}$, e.g. gravitational, electric, or magnetic forces; also apparent forces, such as the centrifugal force in a rotating system, would show up here. The body force $\boldsymbol{F}$ (per volume) is expressed as the product of density and the force per unit mass, hence $\boldsymbol{F} = \rho \boldsymbol{f}$; the force per unit mass $\boldsymbol{f}$ can also be interpreted as an acceleration, for example in the gravitational case it is given by the gravitational acceleration, $\boldsymbol{f} = \boldsymbol{g}$. In summary, the balance equation (2.5) for momentum has the form

$$\frac{\partial \rho \boldsymbol{v}}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v} - \mathbb{P}) - \rho \boldsymbol{f} = 0\,. \tag{2.7}$$

Note that two vectors without an interposed operator form a tensor product, as in the argument of the divergence operator in (2.7).

Energy

Another quantity that is balanced in the form of Eq. (2.5) is the *total energy*, $e^{\mathrm{tot}} = e + k$, the sum of internal energy e and *kinetic energy* $k = (\boldsymbol{v} \cdot \boldsymbol{v})/2$. The mechanical changes are given by the work done on the fluid by external forces $\boldsymbol{F}$ and surface forces $\mathbb{P} \cdot \boldsymbol{n}$.

Additionally, energy may be exchanged across the boundary of a volume in the form of a heat flux $\boldsymbol{q}$. Hence the integral balance for the volume $V(t)$ is

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{V(t)} \rho(e + \tfrac{1}{2}\boldsymbol{v}\cdot\boldsymbol{v})\,\mathrm{d}V = \int\limits_{V(t)} \rho(\boldsymbol{v}\cdot\boldsymbol{f})\,\mathrm{d}V + \int\limits_{\partial V(t)} \boldsymbol{v}\cdot\mathbb{P}\cdot\boldsymbol{n}\,\mathrm{d}(\partial V) - \int\limits_{\partial V(t)} \boldsymbol{q}\cdot\boldsymbol{n}\,\mathrm{d}(\partial V)\,, \tag{2.8}$$

which translates to the differential expression

$$\frac{\partial}{\partial t}(\rho(e+k)) + \nabla\cdot(\rho(e+k)\boldsymbol{v}) - \nabla\cdot(\mathbb{P}\cdot\boldsymbol{v}) + \nabla\cdot\boldsymbol{q} - \rho\boldsymbol{f}\cdot\boldsymbol{v} = 0\,. \tag{2.9}$$

Remark 2.3 *The flux* $\boldsymbol{q}$ *may also concern other energy forms, e.g. radiation. Volumetric energy sources are not considered here.* □

An overview of the conservation equations with the chosen constitutive relations can be found in Section 2.4.

2.3 Constitutive equations for the non-convective transport terms

Regarding Eqs. (2.7) and (2.9), so far the question is open how the non-convective fluxes depend on the state of the fluid. In this section, standard assumptions for the functional dependence of the stress tensor $\mathbb{P}$ and the heat flux vector $\boldsymbol{q}$ are discussed.

2.3.1 Viscous stress tensor: Newtonian fluids

In the balance equations for momentum and energy given in Section 2.2, the *stress tensor* or *pressure tensor* $\mathbb{P}$ has been used to define the non-convective flux of momentum. It contributes to the total momentum flux through a surface element (imaginary or real) in or at the boundary of the fluid.

As has been mentioned previously, the stress tensor is a symmetric rank two tensor.[3] From the physical point of view, $\mathbb{P}$ can be decomposed into a pressure-related part and a contribution by the viscous processes in the fluid: $\mathbb{P} = -p\mathbb{I} + \mathbb{P}_{\mathsf{visc}}$, with the unit tensor $\mathbb{I}$. The pressure p is the thermodynamic pressure in the fluid and its relation to other state variables is subject of Section 2.6. The viscous contribution, on the other hand, depends on material properties of the fluid and on the flow, i.e., the fields of the dynamic variables, particularly the velocity field $\boldsymbol{v}$. In the following, the simplest—namely linear—ansatz for $\mathbb{P}_{\mathsf{visc}}$ in terms of $\boldsymbol{v}$ is described.

Using the *rate of deformation tensor* or *rate of strain tensor*

$$\mathbb{D} := \tfrac{1}{2}(\nabla\boldsymbol{v} + (\nabla\boldsymbol{v})^{\mathsf{T}})\,, \tag{2.10}$$

[3]The symmetry requirement holds for nonpolar fluids.

the most general linear form that satisfies the physical constraints is (Landau and Lifshitz, 1963, p. 48)

$$\mathbb{P}_{\mathsf{visc}} = (\lambda - \tfrac{2}{3}\eta)(\nabla \cdot \boldsymbol{v})\mathbb{I} + 2\eta\mathbb{D}\,, \tag{2.11}$$

involving two positive scalar coefficients: the *dynamic viscosity* or *shear viscosity* η, and the *dilatational viscosity* or *bulk viscosity* λ, which will be further examined below.

Remark 2.4 *The above definition of the two scalar coefficients has to be contrasted with a different possibility that is also often encountered, namely*

$$\mathbb{P}_{\mathsf{visc}} = \tilde{\lambda}(\nabla \cdot \boldsymbol{v})\mathbb{I} + 2\tilde{\eta}\mathbb{D}\,.$$

The coefficients $\tilde{\eta}$ and $\tilde{\lambda}$ are called the first and second coefficients of viscosity, respectively. The splitting in Eq. (2.11)*, however, allows to differentiate between two mathematical components of the tensor $\mathbb{P}_{\mathsf{visc}}$. Noting that every tensor $\mathbb{T}$ can be decomposed into three parts, an isotropic component (which does not change when the frame of reference is rotated), and anisotropic parts (with zero trace), of which one is symmetric and the other one asymmetric, hence*

$$\mathbb{T} = \underbrace{\tfrac{1}{3}\mathbb{I}\cdot\cdot\,\mathbb{T}\,\mathbb{I}}_{\textit{isotropic}} + \underbrace{\overbrace{\tfrac{1}{2}(\mathbb{T} + \mathbb{T}^{\mathsf{T}} - \tfrac{2}{3}\mathbb{I}\cdot\cdot\,\mathbb{T}\,\mathbb{I})}^{\textit{symmetric}} + \overbrace{\tfrac{1}{2}(\mathbb{T} - \mathbb{T}^{\mathsf{T}})}^{\textit{asymmetric}}}_{\textit{anisotropic/traceless}}\,,$$

with the double contraction notation $\mathbb{I}\cdot\cdot\,\mathbb{T}$, one finds that the definition in (2.11) *leaves the*

$$\begin{aligned} &\textit{isotropic part,} && \tfrac{1}{3}\mathbb{I}\cdot\cdot\,\mathbb{P}_{\mathsf{visc}}\,\mathbb{I} && = \lambda\nabla\cdot\boldsymbol{v}\mathbb{I}\,,\\ &\textit{anisotropic symmetric part,} && \tfrac{1}{2}(\mathbb{P}_{\mathsf{visc}} + \mathbb{P}^{\mathsf{T}}_{\mathsf{visc}} - \tfrac{2}{3}\mathbb{I}\cdot\cdot\,\mathbb{P}_{\mathsf{visc}}\,\mathbb{I}) && = -\tfrac{2}{3}\eta(\nabla\cdot\boldsymbol{v})\mathbb{I} + 2\eta\mathbb{D}\,,\end{aligned}$$

of the viscous stress tensor $\mathbb{P}_{\mathsf{visc}}$ governed by just one parameter each. Also, frequently the dilatational viscosity λ may be set to zero, as will be reasoned below. By comparison with (2.11) *the viscous tensor $\mathbb{P}_{\mathsf{visc}}$ is found to be traceless in that case.* □

The importance of the linear ansatz for the dependence of stress on the rate of strain derives from the fact that many technically interesting fluids are modeled quite well by it. Especially most gases and their mixtures, but also liquids with low molecular weight, like water and mineral oils, belong to this class of fluids, which are named *Newtonian fluids*.

Viscosity coefficients

In general, both the dynamic and dilatational viscosity coefficients depend on the thermodynamic state and the composition of the fluid, but by definition *not* on the shear rate, e.g. $\eta = \eta(p,T)$, $\lambda = \lambda(p,T)$. For many materials and flows, the dependence is assumed either to be constant or to have a simple relationship. For instance the Sutherland formula for the dynamic viscosity sets $\eta \sim \sqrt{T}$ and neglects the pressure dependence.

Kinetic gas theory finds the dilatational viscosity λ to be nonzero if kinetic energy of the fluid molecules can be transferred to internal degrees of freedom. The value of λ depends on the characteristic time of the energy transfer and vanishes for monatomic fluids. Often, the dilatational effect is dominated by the shear viscosity, and many studies thus use the *Stokes hypothesis* $\lambda = 0$.

2.3.2 Heat flux density

The non-convective flux of internal energy $\boldsymbol{q}$ is related by thermodynamics to temperature, a quantity that describes the heat of a body or medium. Following Fourier's law of heat conduction—valid for cases in which the temperature gradient in the fluid is small—the heat flux density due to thermal conduction is modeled as

$$\boldsymbol{q} = -\kappa \nabla T \,, \tag{2.12}$$

with the scalar *thermal conductivity* $\kappa(p, T) > 0$. Landau and Lifshitz (1963, p. 187) discuss the applicability of this linear approximation and give values of κ for different materials, see also (Atkins and de Paula, 2002).

2.4 The system of equations in vector notation

In a single phase, the flow is described by the *continuity equation* for mass conservation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0 \,, \tag{2.13a}$$

the equation for the momentum balance, which for Newtonian fluids can be written as

$$\frac{\partial \rho \boldsymbol{v}}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v} + p\mathbb{I} - (\lambda - \tfrac{2}{3}\eta)(\nabla \cdot \boldsymbol{v})\mathbb{I} - 2\eta\mathbb{D}) - \rho \boldsymbol{f} = 0 \,, \tag{2.13b}$$

and an equation that balances the total energy. Also here, the Newtonian assumption is made and the energy is assumed to be either of kinetic (contained in the macroscopic motion of the fluid) or internal nature (held by the internal degrees of freedom of the fluid molecules). With these assumptions the energy equation can be given the form

$$\begin{aligned} &\frac{\partial}{\partial t}(\rho(e+k)) + \nabla \cdot (\rho(e+k)\boldsymbol{v}) \\ &\qquad - \nabla \cdot [(-p\mathbb{I} + (\lambda - \tfrac{2}{3}\eta)(\nabla \cdot \boldsymbol{v})\mathbb{I} + 2\eta\mathbb{D}) \cdot \boldsymbol{v}] - \nabla \cdot (\kappa \nabla T) - \rho \boldsymbol{f} \cdot \boldsymbol{v} = 0 \,. \end{aligned} \tag{2.13c}$$

Remark 2.5 *In the following chapters, the pressure term $\nabla \cdot (p\boldsymbol{v})$ in the energy equation is frequently added to the divergence term of the convective fluxes.* □

2.5 Choice of a coordinate system

It is a fundamental property of the laws of physics that they are independent of the choice of coordinate system. To solve practical problems, however, the differential equations that have been derived so far need to be expressed with respect to a coordinate system. A particular system may be selected based on considerations regarding symmetries of the problem being treated and the complications which may occur in the coordinate equations (e.g., singularities). The numerical techniques developed here are meant for application in general and time-dependent geometries, which may not even be known a priori. Hence exploiting symmetries when choosing the coordinate system is not possible.

Instead, a fixed Cartesian coordinate system is used, which leads to a relatively compact coordinate form of the equations. A position vector $\boldsymbol{x}$ can be represented by the d-tuple of coordinates $(x_1, \ldots, x_d)$ with respect to this Cartesian system. For brevity, when using the coordinate form of the equations, the Einstein summation convention is implied for repeated indices. Summation indices are—where possible—chosen from the range $m, \ldots, s$.[4] Free indices that enumerate different coordinate directions are typically $\bar{\imath}$, $\bar{\jmath}$, or $\bar{k}$. An index with an overbar, like $\bar{\imath}$, concerns the space dimension(s), i.e., $\bar{\imath} = 1, \ldots, d$, see also Chapter 3. With this choice of coordinate system and notation, the Navier–Stokes equations (2.13) take the form

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho v_{\bar{n}})}{\partial x_{\bar{n}}} = 0\,, \tag{2.14a}$$

$$\frac{\partial(\rho v_{\bar{\imath}})}{\partial t} + \frac{\partial(\rho v_{\bar{\imath}} v_{\bar{n}} + \delta_{\bar{\imath}\bar{n}} p)}{\partial x_{\bar{n}}} = \frac{\partial}{\partial x_{\bar{n}}}\left((\lambda - \tfrac{2}{3}\eta)\frac{\partial v_{\bar{m}}}{\partial x_{\bar{m}}}\delta_{\bar{\imath}\bar{n}} + 2\eta D_{\bar{\imath}\bar{n}}\right) + \rho f_{\bar{\imath}}\,, \tag{2.14b}$$

$$\begin{aligned}\frac{\partial(\rho(e+k))}{\partial t} + \frac{\partial((\rho(e+k)+p)v_{\bar{n}})}{\partial x_{\bar{n}}} &= \\ \frac{\partial}{\partial x_{\bar{m}}}\left[\left((\lambda - \tfrac{2}{3}\eta)\frac{\partial v_{\bar{r}}}{\partial x_{\bar{r}}}\delta_{\bar{m}\bar{n}} + 2\eta D_{\bar{m}\bar{n}}\right)v_{\bar{n}}\right] &+ \frac{\partial}{\partial x_{\bar{n}}}\left(\kappa\frac{\partial T}{\partial x_{\bar{n}}}\right) + \rho f_{\bar{n}} v_{\bar{n}}\,.\end{aligned} \tag{2.14c}$$

The Euler flux matrix with the columns F^{e}_j, $j = 0, \ldots, d$, contains the time derivative

[4]Sometimes letters are used as summation indices which have also another meaning, e.g. n is also used as time slab index and to denote a normal vector, and s symbolizes entropy in the treatment of thermodynamical properties. However the meaning can always be inferred from the usage within an equation.

arguments and the flux terms so that $\partial F^{\mathrm{e}}_n/\partial x_n$ represents the left hand side of (2.14):[5]

$$F^{\mathrm{e}} = \left(\begin{array}{c|c} \rho & \rho v_{\bar{\jmath}} \\ \rho v_{\bar{\imath}} & \rho v_{\bar{\imath}} v_{\bar{\jmath}} + p\delta_{\bar{\imath}\bar{\jmath}} \\ \rho(e+k) & v_{\bar{\jmath}}(\rho(e+k)+p) \end{array} \right). \tag{2.15}$$

The corresponding matrix with the diffusive fluxes reads

$$F^{\mathrm{d}} = \left(\begin{array}{c} (0)_{\bar{\jmath}} \\ (\lambda - \frac{2}{3}\eta)\dfrac{\partial v_{\bar{m}}}{\partial x_{\bar{m}}}\delta_{\bar{\imath}\bar{\jmath}} + 2\eta D_{\bar{\imath}\bar{\jmath}} \\ \left((\lambda - \frac{2}{3}\eta)\dfrac{\partial v_{\bar{m}}}{\partial x_{\bar{m}}}\delta_{\bar{\jmath}\bar{n}} + 2\eta D_{\bar{\jmath}\bar{n}}\right) v_{\bar{n}} + \kappa\dfrac{\partial T}{\partial x_{\bar{\jmath}}} \end{array} \right). \tag{2.16}$$

2.6 Thermodynamics

In Section 2.5, the component form of the conservation equations for mass, momentum, and energy has been obtained. On closer inspection, a problem remains: There are $d+2$ *prognostic equations* (i.e., ones that contain a temporal derivative $\partial/\partial t$), but in total $d+4$ unknowns are present in the equations, namely density, d velocity components, pressure, energy, and temperature. To close the system of equations, two additional relations have to be specified. Neglecting temporarily the motion of the fluid,[6] these relations have to describe the interplay of density, pressure, temperature, and internal energy. The description of matter in terms of these variables is the subject of thermodynamics. Standard thermodynamics is described in many textbooks, e.g. (Panton, 1984; Atkins and de Paula, 2002; Zdunkowski and Bott, 2004) and suffices for the goals pursued here. In the remainder of this section, some relations of particular use in the sequel are derived or mentioned. For details the aforementioned references may be consulted.

The total energy $e^{\mathrm{tot}} = e + k$, for which conservation is postulated, is split up in the kinetic part k and a not directly observable part, the internal energy e, which is related to the energy stored in internal degrees of freedom of the molecules. The latter means that the physical effects responsible for the energy content are left unspecified. Consequently the processes involving the conversion of internal energy and its interaction with other variables of state have to be modeled or determined experimentally. From the conservation statement for energy, which is called the *first law of thermodynamics* in this context, it follows that the change de of the internal energy of a system is given by the sum of the

[5]In this notation, the indices $\bar{\imath}$ and $\bar{\jmath}$ are not bound by the left hand side and thus enumerate rows and columns of the matrix, respectively.

[6]Neglecting the velocity dependence effectively means that the thermodynamic state is assumed independent of the dynamic state, except by indirect effects, e.g. through pressure changes in a convergent or divergent mass flux field.

heat (per unit mass) that is added to the system, $đq$, and the work that is performed on it, $đa = -p\,\mathrm{d}\alpha + \mathbb{P}_{\mathsf{visc}} \cdot \mathrm{d}\boldsymbol{f}$,

$$\mathrm{d}e = đq + đa\,. \tag{2.17}$$

Both energy forms on the right hand side constitute inexact differentials $đ$, i.e., the value of an integral like $\int_{z_1}^{z_2} đq$ not only depends on the initial and end state, z_1 and z_2, respectively, but also on the path (or in physical terms: the process) that is used for the integration. This is in contrast to the variables of state—like internal energy, density, and pressure—whose value only depends on the state z and not on the integration path or process. Equation (2.17) is written in intensive form so that it is also applicable to systems that exchange mass with their surroundings. Considering the internal energy e in terms of the state variables α and T leads to the introduction of the *specific heat at constant volume*,

$$c_{\mathsf{v}} := \left(\frac{\partial e}{\partial T}\right)_{\alpha} = \frac{đ}{\mathrm{d}T}(q + a)\,. \tag{2.18}$$

Denoting the viscous contribution to the energy balance as $đw = \mathbb{P}_{\mathsf{visc}} \cdot \mathrm{d}\boldsymbol{f}$, the energy balance can be written as

$$\mathrm{d}e + p\,\mathrm{d}\alpha = đ(q + w)\,. \tag{2.19}$$

Applying the Legendre-transformation

$$\mathrm{d}e + p\,\mathrm{d}\alpha = \mathrm{d}(e + p\,\alpha) - \alpha\,\mathrm{d}p\,, \tag{2.20}$$

on the right hand side, the *specific enthalpy* is introduced as

$$h = e + p\,\alpha\,, \tag{2.21}$$

which measures the energy exchange of a system at constant pressure. Analogously to the internal energy case, this leads to the definition of the *specific heat at constant pressure*,

$$c_{\mathsf{p}} := \left(\frac{\partial h}{\partial T}\right)_{p} = \frac{đ}{\mathrm{d}T}(q + w)\,. \tag{2.22}$$

The introduction of entropy s and the second law of thermodynamics are omitted here and the previously cited references should be consulted for details. The fundamental differential equation of thermodynamics is

$$T\,\mathrm{d}s = \mathrm{d}e + p\,\mathrm{d}\alpha\,. \tag{2.23}$$

In combination with the specific entropy s, additional state functions can be introduced. For Section 2.8.2, in particular the *chemical potential* $\mu = h - T s$ is of interest. It is a potential in terms of the variables p and T, which is expressed by the de Donder statements,

$$\left(\frac{\partial \mu}{\partial T}\right)_p = -s\,, \qquad \left(\frac{\partial \mu}{\partial p}\right)_T = \alpha\,. \tag{2.24}$$

Another way to characterize different fluids, especially regarding their compressibility, is given by two parameters that are defined as the relative changes of the specific volume with temperature and pressure, respectively,

$$\textit{isobaric expansion coefficient,} \quad \alpha_p := \frac{1}{\alpha}\left(\frac{\partial \alpha}{\partial T}\right)_p\,, \tag{2.25a}$$

$$\textit{isothermal compressibility,} \quad \beta_T := -\frac{1}{\alpha}\left(\frac{\partial \alpha}{\partial p}\right)_T\,. \tag{2.25b}$$

Both are material coefficients and in general depend on the thermodynamical state. Knowledge of these two quantities and either c_p or c_v in terms of the state allows to work out the previously mentioned equations of state, too, see, e.g., (Polner, 2005). In case $\alpha_p = 0$, then the variation of the specific volume at constant pressure vanishes, $(\delta\alpha)_p = 0$, which is called *temperature incompressible*. If $\beta_T = 0$, then the specific volume is independent of pressure (at constant temperature), $(\delta\alpha)_T = 0$, and the material is *pressure incompressible*. If both coefficients are zero, the material is called *incompressible*.

Many thermodynamical properties of a fluid can be computed when a small number of quantities is known. In the following, the three state functions α_p, β_T, and c_p are assumed given (from measurements or theoretical considerations). As an example of the thermodynamic calculations needed in Section 2.8 and Appendix A, the derivatives $(\partial e/\partial T)_p$ and $(\partial e/\partial p)_T$, are related to the three known quantities. Starting from the chemical potential $\mu = e + p\alpha - Ts$, whose natural coordinates are p and T, one finds from (2.24):

$$-s = \left(\frac{\partial \mu}{\partial T}\right)_p = \left(\frac{\partial e}{\partial T}\right)_p + p\left(\frac{\partial \alpha}{\partial T}\right)_p - s - T\left(\frac{\partial s}{\partial T}\right)_p \tag{2.26}$$

$$\Rightarrow \quad \left(\frac{\partial e}{\partial T}\right)_p = -p\left(\frac{\partial \alpha}{\partial T}\right)_p + T\left(\frac{\partial s}{\partial T}\right)_p = -p\alpha\alpha_p + c_\mathsf{p}\,, \tag{2.27}$$

where $(\partial s/\partial T)_p = c_\mathsf{p}/T$, which follows from (2.22) and (2.23), has been substituted. Further, by using the Kelvin–de Donder relation $(\partial s/\partial p)_T = -(\partial \alpha/\partial T)_p$, see, for example, (Zdunkowski and Bott, 2004), the derivative with respect to pressure is found from

$$\alpha = \left(\frac{\partial \mu}{\partial p}\right)_T = \left(\frac{\partial e}{\partial p}\right)_T + \alpha + p\left(\frac{\partial \alpha}{\partial p}\right)_T - T\left(\frac{\partial s}{\partial p}\right)_T \tag{2.28}$$

$$\Rightarrow \quad \left(\frac{\partial e}{\partial p}\right)_T = -p\left(\frac{\partial \alpha}{\partial p}\right)_T + T\left(\frac{\partial s}{\partial p}\right)_T = p\alpha\beta_T - T\alpha\alpha_p\,. \tag{2.29}$$

Having introduced these general relations, some of the mentioned thermodynamic functions need to be specified to include the behavior of a given fluid in the prognostic equations (2.14).

2.7 Thermodynamical models for fluids

In the previous section, several quantities have been introduced that characterize the thermodynamical state of a substance. The problem that arose earlier was to determine how the state of a fluid changes given a certain stimulus, for example a modification of density or internal energy—cf. Eqs. (2.14a) and (2.14c). A solution to this problem is often provided in the form of *equations of state* (EOS). Such equations relate the thermodynamic quantities with each other. For the current purposes, a description with two equations of state is suitable: For each medium, a caloric EOS for the internal energy is given in terms of the specific volume and temperature, $e = e(\alpha, T)$, and a thermal EOS relates pressure, specific volume, and temperature, for example in the form $p = p(\alpha, T)$. Given the thermodynamical theory from Section 2.6, these two would be sufficient to determine all other relations, but for ease of reference an additional equation is given for entropy. All mentioned relations depend on the medium considered. The following sections contain the equations of state of three important fluid models.

2.7.1 Ideal gas

When the volume of the molecules of a gas is negligible and the potential energy of the their interaction is small compared to the sum of the molecules' kinetic energies,[7] then the substance is called an *ideal gas*. For the purposes pursued here, an ideal gas is assumed to fulfill the following equations of state:

$$e(T) = e_0 + c_\mathrm{v}(T - T_0)\,, \tag{2.30a}$$

$$p(\alpha, T) = \frac{RT}{\alpha}\,, \tag{2.30b}$$

where R is the *gas constant*, and c_v is the specific heat at constant volume, which has additionally been assumed constant. The entropy of an ideal gas is given by[8]

$$s(\alpha, T) = s_0 + R \ln \frac{\alpha}{\alpha_0} + c_\mathrm{v} \ln \frac{T}{T_0}\,. \tag{2.31}$$

[7]Obviously the potential of the interaction cannot be zero, for otherwise there would be no interaction between the molecules at all.

[8]Note that the temperature T_0 in (2.31) is not necessarily the same as in (2.30a). For each equation, T_0 (and α_0) belongs to the state at which the entropy (energy) attains the value s_0 (e_0). For Eq. (2.30a), the base temperature is often set to $T_0 = 0$, which is not possible in (2.31). If a single state (p_0, α_0, T_0) is chosen, then it must fulfill (2.30b).

For the compressibility parameters defined in (2.25) one finds

$$\alpha_p = \frac{1}{T}, \quad \text{and} \quad \beta_T = \frac{1}{p}. \tag{2.32}$$

The sound speed a in an ideal gas is given by

$$a = \sqrt{\left(\frac{\partial p}{\partial \rho}\right)_s} = \sqrt{\frac{\gamma \alpha}{\beta_T}} = \sqrt{\frac{\gamma p}{\rho}} = \sqrt{\gamma R T}, \tag{2.33}$$

with the *adiabatic coefficient* $\gamma = c_\mathsf{p}/c_\mathsf{v}$, the ratio of the specific heats at constant pressure and volume.

2.7.2 Van der Waals gas

In Section 2.7.1, an ideal gas was defined by the negligible volume of the molecules of a gas and the low degree of their interaction. At low densities this idealization holds for all gases, but at higher densities the interaction of the molecules may change the behavior of the gas. Other models have been developed that take into account the finite volume occupied by the molecules (*covolume EOS*) and additionally the attraction of the molecules, for example the *van der Waals EOS*.

The assumptions underlying the van der Waals (vdW) EOS are (i) that the molecules are finite-sized impenetrable spheres whose specific volume b reduces the available volume for the expansion of the gas from α to $\alpha - b$, and (ii) the movement of the molecules is modified by the attractive forces, which change the pressure through the frequency and force of molecular collisions and are modeled by a quadratic term in density with proportionality constant a. See for example (Atkins and de Paula, 2002) for a justification and values of the parameters a and b. The equations of state read:

$$e(\alpha, T) = e_0 + c_\mathsf{v}(T - T_0) + a\left(\frac{1}{\alpha_0} - \frac{1}{\alpha}\right), \tag{2.34a}$$

$$p(\alpha, T) = \frac{RT}{\alpha - b} - \frac{a}{\alpha^2}, \tag{2.34b}$$

where again c_v has been assumed constant. Note that in this case, the specific heat at constant pressure cannot be constant, as follows from

$$c_\mathsf{p} - c_\mathsf{v} = \frac{R^2 \alpha^3 T}{R\alpha^3 T - 2a(\alpha - b)^2}. \tag{2.35}$$

The derivation of the previous relationship as well as of the formulae for the compressibili-

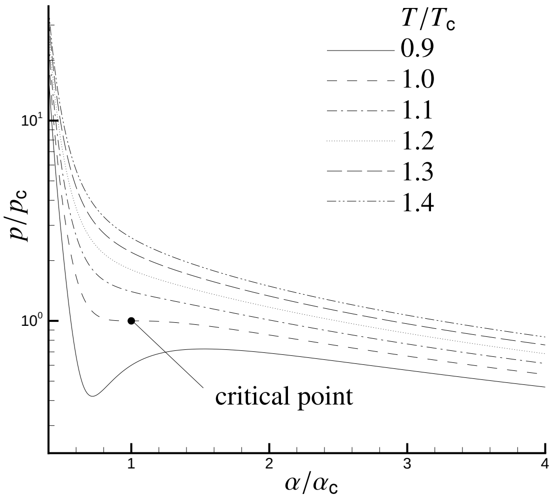


Figure 2.1: Pressure of a van der Waals gas as a function of the specific volume at different temperatures. All quantities in the diagram are scaled by the values of the state coordinates in the critical point.

ties can be found in (Polner, 2005),

$$\alpha_p = \frac{1}{\frac{\alpha}{\alpha-b}T - 2a\frac{\alpha-b}{R\alpha^2}}, \quad \text{and} \quad \beta_T = \frac{1}{\frac{\alpha}{\alpha-b}p - \frac{a}{\alpha^2}\left(2 - \frac{\alpha}{\alpha-b}\right)}. \tag{2.36}$$

The entropy of the vdW gas is

$$s(\alpha, T) = s_0 + R\ln\frac{\alpha-b}{\alpha_0-b} + c_v\ln\frac{T}{T_0}. \tag{2.37}$$

Naturally, the two additional parameters in the vdW EOS allow to better capture the behavior of (real) gases. Further, the EOS reproduces the ideal gas model for $a = 0$, $b = 0$ and at sufficiently high temperatures or low densities. In general, however, the van der Waals gas is substantially more complicated as the pressure equation of state (2.34b) is a rational function in p, α, and T, and computing arbitrary states is made difficult by the nonlinearity. For temperatures below a critical value T_c, the (α, p)-graph is non-monotonic in a neighborhood of the *critical point* (α_c, p_c), cf. Figure 2.1, which makes sensible state evaluations impossible in that range. Consequently, the (iterative) computation of thermodynamical parameters fails close to the critical point. This deficiency can be remedied, e.g. with the Maxwell construction. In this work only cases with states away from the critical point are treated, so that the monotonicity of the state functions is guaranteed.

2.7.3 Other models for compressible fluids

The modeling of fluids has gone far beyond the two previously presented examples. The immense broadening of theoretical understanding, experimental and measurement capabilities, and specialized technical applications have contributed to the large diversity of equations of state that are presently available to describe (compressible) fluids. Many references to relevant literature can be found in (Sengers, 2000). The principle that the equations relate the state variables as in (2.30) or (2.34) remains the same, but often the equations become more complicated than in the cited examples. Unfortunately, many of these EOS apply only in a certain region of the thermodynamical state space. A manifestation of such a restriction has already surfaced in Section 2.7.2: The lack of monotonicity at temperatures below the critical value T_c is physically questionable and doomed to mathematical complications. In the later sections of this chapter, and equally in the final numerical method, equations of state have to be inverted to convert between different variables. The inversion is not generally possible if an EOS is non-monotonic. None of the more involved fluid models will be detailed here but their usage in the developed formalism is possible provided they are applied in a subset of the state space where the thermodynamical functions are defined unambiguously.

A fundamentally different fluid model from those described so far is the incompressible fluid. It is of comparable importance as the ideal gas because it delivers a widely applicable description that represents the essential physics of many processes: Flows in which the pressure differences are not sufficiently large to lead to a significant compression of the fluid are usually considered with the incompressible fluid model.

2.7.4 Incompressible fluid

Incompressibility is an idealization that can—from the theoretical point of view—not be attained, but it is a useful assumption for modeling many physical processes. In the thermodynamical framework, the compressibility coefficients are both zero, $\alpha_p = 0, \beta_T = 0$. Moreover, the pressure loses its thermodynamic meaning and becomes a purely mechanical variable, cf. (Panton, 1984); hence no EOS for the pressure exists. Due to the constant density, the internal energy depends only on temperature, and, assuming again a constant specific heat at constant volume, c_v, the energy is given by

$$e(T) = e_0 + c_\mathsf{v}(T - T_0)\,. \tag{2.38}$$

For incompressible fluids the specific heats at constant volume and at constant pressure are equal, $c_\mathsf{v} = c_\mathsf{p}$, and hence their ratio, the adiabatic coefficient, is $\gamma = 1$. The entropy is given by

$$s(T) = s_0 + c_\mathsf{v} \ln \frac{T}{T_0}\,. \tag{2.39}$$

The incompressible Navier–Stokes equations

Assuming incompressibility and homogeneity of the fluid, the Navier–Stokes equations can be written without any reference to density at all:

$$\nabla \cdot \boldsymbol{v} = 0\,, \tag{2.40a}$$

$$\frac{\partial \boldsymbol{v}}{\partial t} + \nabla \cdot (\boldsymbol{v}\boldsymbol{v} + \tilde{p}\mathbb{I}) = \nu \Delta \boldsymbol{v}\,, \tag{2.40b}$$

with the *kinematic viscosity* $\nu = \eta/\rho$ and the pressure given analogously by $\tilde{p} = p/\rho$. The energy equation can be converted to a prognostic equation for temperature. The transformed relations are, however, not used further, as the goal of the present work is to develop a numerical tool that applies to both compressible and incompressible flow.

Having derived the equations for fluid flow and formulated the equations of state for different fluids a closed system of equations is available. The next section takes a look at this system from a different angle.

2.8 The Navier–Stokes equations in terms of different sets of variables

Starting from the consideration of conservation laws in Section 2.2, the Navier–Stokes equations for fluid flow have been derived, and they were given in Cartesian coordinate form in Section 2.5. The system of partial differential equations was closed by adding two equations of state, which characterize the thermodynamical properties of the considered fluid. It is important to realize that, just like the possibility to pick different coordinate systems, the set of *variables* that is used to find a solution to the Navier–Stokes equations in space and time is *not* an intrinsic property of the set of equations. When deriving the equations, conserved quantities were balanced, and it may seem most natural to treat these as the variables: the densities of mass ρ, the momentum components $\rho v_{\bar{\imath}}$, and total energy ρe^{tot}.[9] However, it has already been shown that in some cases, e.g. due to additional knowledge about specific physical properties, a different set of variables may be more beneficial to use, see the equations for incompressible media above.[10] Also in other contexts replacing some of the variables by others (e.g., energy by enthalpy) may expose or exploit special properties and possibly make the system easier to solve.

[9]As mentioned previously, additional unknowns are included in the PDE system and have to be related: temperature T and pressure p.

[10]Note that in the case of the incompressible Navier–Stokes equations not only the variables are changed, so that typically momentum components are replaced by velocities $v_{\bar{\imath}}$ and temperature takes the place of total specific energy, but also the equations are modified, though this concerns mainly the division by a constant.

In the following paragraphs, transformations to different variable sets are treated, but in contrast to the examples mentioned so far, these transformations primarily aim at mathematical targets rather than at exploiting special physical knowledge. Parts of the analysis of the transformed systems rely on the quasi-linear form of the Navier–Stokes equations, which is discussed in Section 2.8.1. Section 2.8.2 introduces different variable sets and highlights their properties.

2.8.1 Quasi-linear form of the Navier–Stokes equations

For conservation variables $U := [\rho, \rho v_{\bar{\imath}}, \rho e^{\mathrm{tot}}]$, the Navier–Stokes equations (2.14) can be written in the form

$$\frac{\partial U}{\partial t} + \frac{\partial F^{\mathrm{e}}_{\bar{n}}}{\partial x_{\bar{n}}} = \frac{\partial F^{\mathrm{d}}_{\bar{n}}}{\partial x_{\bar{n}}} + S \,, \tag{2.41}$$

with $F^{\mathrm{e}}_{\bar{\imath}}$ the Euler and $F^{\mathrm{d}}_{\bar{\imath}}$ the diffusive flux in the $x_{\bar{\imath}}$-coordinate direction, and possible source terms included in S. To obtain the quasi-linear form of this equation use will be made of the notation $A^U_{\bar{\imath}} = \partial F^{\mathrm{e}}_{\bar{\imath}}/\partial U$ for the Jacobians of the Euler flux in the $x_{\bar{\imath}}$-coordinate direction (including the pressure term). The diffusive fluxes in the Navier–Stokes equations for Newtonian fluids depend only on the first order derivatives of the velocities, cf. Section 2.3.1, which is exploited by the exact relation $F^{\mathrm{d}}_{\bar{\imath}} = K^U_{\bar{\imath}\bar{n}} \partial U/\partial x_{\bar{n}}$. With these definitions, Equation (2.41) can be rewritten as

$$\frac{\partial U}{\partial t} + A^U_{\bar{n}} \frac{\partial U}{\partial x_{\bar{n}}} = \frac{\partial}{\partial x_{\bar{m}}} \left(K^U_{\bar{m}\bar{n}} \frac{\partial U}{\partial x_{\bar{n}}} \right) + S \,, \tag{2.42}$$

which is the starting point for introducing a different set of variables V. In the quasi-linear context, the Jacobian of the transformation $U(V)$, i.e., $A^V_0 = \partial U/\partial V$, is inserted in front of every partial derivative of U, which is then replaced by a partial derivative of V,

$$A^V_0 \frac{\partial V}{\partial t} + A^V_{\bar{n}} \frac{\partial V}{\partial x_{\bar{n}}} = \frac{\partial}{\partial x_{\bar{m}}} \left(K^V_{\bar{m}\bar{n}} \frac{\partial V}{\partial x_{\bar{n}}} \right) + S \,, \tag{2.43}$$

with $A^V_{\bar{\imath}} = A^U_{\bar{\imath}}\, \partial U/\partial V$ and $K^V_{\bar{\imath}\bar{\jmath}} = K^U_{\bar{\imath}\bar{\jmath}}\, \partial U/\partial V$.

Remark 2.6 *The mapping between U and V must be bijective, which, in symmetrization theory, is ensured by requiring $\partial U/\partial V$ to be positive definite (Barth, 1999).* □

Finally, it should be emphasized that using a different variable set is also possible in the original nonlinear set of equations (2.41) by interpreting the functional dependence on U as $U(V)$. The numerical methods derived in Chapters 4 and 5 take this route and solve the nonlinear equations. The quasi-linear form is, on the one hand, the starting point for numerical analysis, and on the other hand the linearization can be used to solve the nonlinear system with a Newton-like method, which will be expounded in Chapter 4.

The question that remains is, obviously, which alternative variable sets V are suitable for solving the Navier–Stokes equations. The following section contains an overview of several possibilities. Note that the letter V, which so far signifies a generic variable set, symbolizes a specific set there. In many derivations later on, V is used in the generic sense again, unless its meaning is explicitly restricted.

2.8.2 Sets of variables for the Navier–Stokes equations

Some variable sets that may be used in the Navier–Stokes equations are

$$\text{conservation variables,}\quad U := \left(\rho, \rho v_{\bar{\imath}}, \rho e^{\mathrm{tot}}\right)^{\mathsf{T}} , \tag{2.44}$$

$$\text{primitive variables with density,}\quad Y' := (\rho, v_{\bar{\imath}}, T)^{\mathsf{T}} , \tag{2.45}$$

$$\text{primitive variables with pressure,}\quad Y := (p, v_{\bar{\imath}}, T)^{\mathsf{T}} , \tag{2.46}$$

$$\text{entropy variables,}\quad V := \frac{1}{T}\left(\mu - \frac{1}{2}v_{\bar{n}}^2, v_{\bar{\imath}}, -1\right)^{\mathsf{T}} . \tag{2.47}$$

The total energy e^{tot} is, as previously mentioned, assumed to consist of internal energy e, and kinetic energy $k := v_{\bar{n}}^2/2$, i.e., $e^{\mathrm{tot}} = e + v_{\bar{n}}^2/2$.

Having stated the different variable sets one may ask which consequences the selection of a specific set has for the numerical method to be derived. An overview of previous findings regarding mathematical properties and practicality is given next.

Conservation variables

Many numerical methods for compressible fluids are based on conservation variables, presumably because of their natural occurrence when deriving the conservation laws and because it is easy to derive numerical discretizations based on them. On the other hand, conservation variables are not suitable for computing incompressible flows, which is apparent because the Jacobians $A^U_{\bar{\imath}}$ are not well-behaved in the incompressible limit: some of their entries tend to infinity or to the (undefined) quotient $0/0$, see (Hauke and Hughes, 1998, p. 7). Compared with the other variable sets, Hauke and Hughes (1998, p. 34) find the accuracy to be mediocre (deteriorating for nearly incompressible flow) and systematic errors occur in regions of outflow boundary layers.

Primitive variables with density

The same drawbacks inherent to conservation variables (because of density being included) also apply in this case. In particular, the incompressible limit is not well-behaved, which is why this variable set is discarded from further investigations in the current work.

Primitive variables with pressure

When pressure primitive variables are used then the incompressible limit of the Navier–Stokes equations is well-defined. Additionally, implementations using the Jacobians $A^Y_{\bar{\imath}}$

are more efficient compared to other variable sets, because of the sparseness and simple form of the matrices, see p. 138. Hauke and Hughes (1998, p. 34) find that the primitive variables lack some of the robustness of other variable sets.

Entropy variables

Entropy variables have several intriguing properties of both mathematical and physical relevance. First, they have the well-defined incompressible limit in common with the pressure primitive variable set. They are thus applicable to a wide range of physical conditions, a property that can be preserved by suitably designed numerical methods. Unlike the Jacobians for pressure primitive variables, those for entropy variables, A_i^V, are full matrices, but they have the additional property of being symmetric, see p. 140. The matrix A_0^V is even symmetric positive definite, which degenerates to symmetric positive semi-definiteness in the incompressible case. The symmetrization property for the system (2.43) is actually the origin of the definition of entropy variables, see, e.g., (Barth, 1999; Tadmor, 2003). Also, the matrix $K^V = [K_{\bar{\imath}\bar{\jmath}}^V]_{\bar{\imath}\bar{\jmath}}$ is symmetric positive semidefinite. As Hughes et al. (1986, p. 231) show, an interesting consequence of the definition of the entropy variables is that certain classes of finite element methods based on them satisfy the Clausius–Duhem inequality, i.e., they fulfill the entropy production inequality and thus converge to the physically correct entropy solution. Additional advantages of symmetrized forms in general include global energy stability of Galerkin least-squares and discontinuous Galerkin methods, dimensional consistency of the inner product of test and trial functions in the weak form, and a systematic approach to eigenvector scaling (Barth, 1999, p. 198). Also regarding robustness and the accuracy of some derived quantities, entropy variables have been found favorable (Hauke and Hughes, 1998, p. 34).
The theoretical and practical advantages of entropy variable formulations are opposed by the complexity of using them, which may also be the reason why they have not been exploited more often. The derivation of the variable transformations requires a number of thermodynamical relations and knowledge about the thermodynamical behavior of the considered fluid, see Appendix A. Furthermore, the relationship between entropy and conservation variables is nonlinear, which not only makes the computation more costly but might also incur loss of accuracy when transforming back and forth. Finally, standard boundary conditions may become cumbersome to impose, depending on the numerical method.

In conclusion, the same formulation of the equations may be used for compressible and incompressible flows provided one solves for pressure primitive or entropy variables (Hauke and Hughes, 1998). The numerical method, however, has to maintain this property by avoiding to exploit special cases (e.g., by using an ideal gas numerical flux). When necessary it has to make use of concepts suitable for both flow types, for instance through stabilization with suitably defined operators. The development of such methods is the goal of Chapters 4 and 5. The Jacobians of the variable transformations and fluxes for pressure primitive and entropy variables are summarized in Appendix A.

2.9 Dimensional analysis

For the problem of flow of a fluid with heat conduction as described in Section 2.5, Table 2.2a on page 32 shows a collection of relevant variables that have been identified in the conservation statements and the thermodynamical considerations. For each variable, its dimension with respect to the *fundamental quantities* (or *primary quantities*) length $\mathcal{L}$, mass $\mathcal{M}$, time $\mathcal{T}$, and temperature $\mathcal{W}$ is given. These quantities are considered unrelated and the column rank of the dimension matrix is four, which shows that all primary quantities are needed. Reference values or *units* for the four magnitudes are chosen based on representative values of length L_r, density ρ_r, velocity v_r, and temperature T_r.

Dimensionless groups are found by diagonalizing the dimensional matrix of Table 2.2a, yielding Table 2.2b on page 32, which is now discussed in more detail. In the first block of the table, the chosen reference values for the fundamental quantities are listed; these can be regarded as defining a set of units for measuring physical quantities. Hence, every quantity q can be written as the product of a dimensionless number, q^*, which gives the multiplicity of the second factor, the reference value q_r, i.e., $q = q^* q_\mathrm{r}$. The measurement system with respect to these reference values can be used to measure any quantity whose dimensional formula involves no other fundamental quantities besides those four introduced above. It is customary, however, to introduce additional scales for certain quantities. In this way one can scale the dimensionless quantity to be on the order of magnitude one.[11] For every additional scale that is introduced, a dimensionless conversion coefficient with respect to the scale defined by the reference quantities has to be taken into account. As an example, a scale $\eta_{\mathrm{r}'}$ for the dynamic viscosity η of the fluid is chosen. From Table 2.2b one finds that the scale implied by the four fundamental quantities is $\eta_\mathrm{r} = L_\mathrm{r}\rho_\mathrm{r}v_\mathrm{r}$. By defining $\mathrm{Re}_\eta := \eta_\mathrm{r}/\eta_{\mathrm{r}'}$, the dynamic viscosity can be expressed as $\eta = \eta'\eta_{\mathrm{r}'} = \eta'\mathrm{Re}_\eta^{-1}\eta_\mathrm{r}$. Thereby additional reference values are introduced for the physical parameters λ, η, c_p, c_v, and κ in the second block of Table 2.2b and the following dimensionless coefficients are introduced:

$$\mathrm{Re}_\lambda := \frac{\rho_\mathrm{r} v_\mathrm{r} L_\mathrm{r}}{\lambda_{\mathrm{r}'}}, \tag{2.48}$$

$$\text{Reynolds number,} \quad \mathrm{Re}_\eta := \frac{\rho_\mathrm{r} v_\mathrm{r} L_\mathrm{r}}{\eta_{\mathrm{r}'}}, \tag{2.49}$$

$$\text{Prandtl number,} \quad \mathrm{Pr} := \frac{\eta_{\mathrm{r}'} c_{\mathrm{p}_{\mathrm{r}'}}}{\kappa_{\mathrm{r}'}}, \tag{2.50}$$

[11] Obviously this kind of scaling is only possible if the physical quantity q does not vary over more than one order of magnitude.

$$\text{Eckert number,}^{12} \quad \mathrm{Ec} := \frac{v_\mathrm{r}^2}{c_{\mathrm{p}_{\mathrm{r}'}} T_\mathrm{r}}, \tag{2.51}$$

$$\text{adiabatic coefficient,} \quad \gamma := \frac{c_{\mathrm{p}_{\mathrm{r}'}}}{c_{\mathrm{v}\mathrm{r}'}}. \tag{2.52}$$

These dimensionless numbers show up when the equations are non-dimensionalized and they often act as weights on terms which represent different physical processes. They are necessary because the dimensionless quantities q' have undergone an additional, arbitrary scaling.

The goal of the non-dimensionalization is to remove all references to (arbitrary) scales from the equations and to describe the problem entirely in terms of dimensionless values. However, the typical goal of a simulation is to reproduce a physical configuration. Consequently there has to be a procedure to convert between a description using physical values and the dimensionless formulation. This transformation is evident from Table 2.3:

- Starting from a set of physical values, define both the basic reference quantities $\{L_\mathrm{r}, \rho_\mathrm{r}, v_\mathrm{r}, T_\mathrm{r}\}$, and the additional ones $\{\lambda_{\mathrm{r}'}, \eta_{\mathrm{r}'}, c_{\mathrm{p}_{\mathrm{r}'}}, c_{\mathrm{v}\mathrm{r}'}, \kappa_{\mathrm{r}'}\}$.
  It is important to note that the way these quantities are defined matters in so far as a later re-scaling of a dimensionless result has to use the same method.[13]
  Given the above values, also the dimensionless numbers (2.48)–(2.52) can be evaluated, and all physical values that define the problem can be divided by their reference quantities to obtain the dimensionless values used in the non-dimensionalized equations, e.g., $L^*, \rho^*, v^*, T^*, p^*, e^*, \lambda', \eta', c_\mathrm{p}', c_\mathrm{v}', \kappa'$.

- To scale dimensionless results to physical values, for example the result of a simulation using the non-dimensionalized equations, the dimensionless coefficients q^* or q' have to be known along with the values of the dimensionless numbers (2.48)–(2.52). By assigning the four reference values $\{L_\mathrm{r}, \rho_\mathrm{r}, v_\mathrm{r}, T_\mathrm{r}\}$, all derived reference values are set (cf. Table 2.3) and the physical quantities can be computed.

2.10 Non-dimensionalization of the equations

All physically correct equations have to be dimensionally homogeneous, hence all summands on both sides of such an equation have to be of the same dimension. All previously derived equations fulfill this criterion. In the light of the dimension considerations, every quantity in the equations is written as a dimensionless factor q^* multiplied with the necessary reference value $q_\mathrm{r}(L_\mathrm{r}, \rho_\mathrm{r}, v_\mathrm{r}, T_\mathrm{r})$. Subsequently, the equation is divided by the product

[12] The reference temperature in the Eckert number frequently represents a characteristic temperature *difference* of the flow, $T_\mathrm{r} = \Delta T$

[13] The flow around a sphere is a simple example to elucidate this: while some authors define the length L_r to be the diameter of the sphere, others prefer the radius. Both choices are valid, but the resulting Reynolds numbers differ by a factor of two.

		$\mathcal{L}$	$\mathcal{M}$	$\mathcal{T}$	$\mathcal{W}$
ρ	density	-3	1	0	0
$v_{\bar{\imath}}$	velocity components ($\bar{\imath} = 1, \ldots, d$)	1	0	-1	0
λ	dilatational viscosity	-1	1	-1	0
η	dynamic viscosity	-1	1	-1	0
c_p	specific heat at constant pressure	2	0	-2	-1
c_v	specific heat at constant volume	2	0	-2	-1
T	temperature	0	0	0	1
κ	heat conductivity	1	1	-3	-1
p	pressure	-1	1	-2	0
e	internal energy	2	0	-2	0
α_p	isobaric expansion coefficient	0	0	0	-1
β_T	isothermal compressibility	1	-1	2	0
R	gas constant (ideal gas)	2	0	-2	-1

(a) Physically relevant variables and parameters for the considered problem and their dimension with respect to the fundamental quantities.

	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{T}$	$\mathcal{W}$
L_r	1	0	0	0
$\rho_\mathsf{r} L_\mathsf{r}^3$	0	1	0	0
$v_\mathsf{r}^{-1} L_\mathsf{r}$	0	0	1	0
T_r	0	0	0	1
$\lambda L_\mathsf{r}^{-1} \rho_\mathsf{r}^{-1} v_\mathsf{r}^{-1}$	0	0	0	0
$\eta L_\mathsf{r}^{-1} \rho_\mathsf{r}^{-1} v_\mathsf{r}^{-1}$	0	0	0	0
$c_\mathsf{p} T_\mathsf{r} v_\mathsf{r}^{-2}$	0	0	0	0
$c_\mathsf{v} T_\mathsf{r} v_\mathsf{r}^{-2}$	0	0	0	0
$\kappa L_\mathsf{r}^{-1} \rho_\mathsf{r}^{-1} v_\mathsf{r}^{-3} T$	0	0	0	0
$p \rho_\mathsf{r}^{-1} v_\mathsf{r}^{-2}$	0	0	0	0
$e v_\mathsf{r}^{-2}$	0	0	0	0
$\alpha_p T_\mathsf{r}$	0	0	0	0
$\beta_T \rho_\mathsf{r} v_\mathsf{r}^2$	0	0	0	0
$R T_\mathsf{r} v_\mathsf{r}^{-2}$	0	0	0	0

(b) Units for the fundamental magnitudes have been chosen (top block of the table) and using these, dimensionless variables result from the diagonalization of the table (middle and lower part). For the quantities in the middle block additional reference values are chosen, leading to the introduction of dimensionless constants.

Table 2.2: Dimensions of the physically relevant quantities with respect to the fundamental quantities length $\mathcal{L}$, mass $\mathcal{M}$, time $\mathcal{T}$, and temperature $\mathcal{W}$. Based on the dimensional formulae, dimensionless products are formed.

quantity	used in dimensionless form	reference value		
x	x^*	L_r		
ρ	ρ^*	ρ_r		
$v_{\bar{\imath}}$	$v_{\bar{\imath}}^*$ $(\bar{\imath} = 1, \ldots, d)$	v_r		
T	T^*	T_r		
λ	λ'	$\lambda_{\mathsf{r}'}$	$=$	$\mathrm{Re}_\lambda^{-1} \rho_\mathsf{r} v_\mathsf{r} L_\mathsf{r}$
η	η'	$\eta_{\mathsf{r}'}$	$=$	$\mathrm{Re}_\eta^{-1} \rho_\mathsf{r} v_\mathsf{r} L_\mathsf{r}$
c_p	c'_p	$c_{\mathsf{p}\mathsf{r}'}$	$=$	$\mathrm{Ec}^{-1} v_\mathsf{r}^2 T_\mathsf{r}^{-1}$
c_v	c'_v	$c_{\mathsf{v}\mathsf{r}'}$	$=$	$\gamma^{-1} \mathrm{Ec}^{-1} v_\mathsf{r}^2 T_\mathsf{r}^{-1}$
κ	κ'	$\kappa_{\mathsf{r}'}$	$=$	$\mathrm{Re}_\eta^{-1} \mathrm{Ec}^{-1}\, \mathrm{Pr}^{-1}\, \rho_\mathsf{r} L_\mathsf{r} v_\mathsf{r}^3 T_\mathsf{r}^{-1}$
p	p^*	p_r	$=$	$\rho_\mathsf{r} v_\mathsf{r}^2$
e	e^*	e_r	$=$	v_r^2
α_p	${\alpha_p}^*$	${\alpha_p}_\mathsf{r}$	$=$	T_r^{-1}
β_T	${\beta_T}^*$	${\beta_T}_\mathsf{r}$	$=$	$\rho_\mathsf{r}^{-1} v_\mathsf{r}^{-2}$
R	R^*	R_r	$=$	$v_\mathsf{r}^2 T_\mathsf{r}^{-1}$
s	s^*	s_r	$=$	$v_\mathsf{r}^2 T_\mathsf{r}^{-1}$
V_1	${V_1}^*$	${V_1}_\mathsf{r}$	$=$	$v_\mathsf{r}^2 T_\mathsf{r}^{-1}$
$V_{1+\bar{\imath}}$	${V_{1+\bar{\imath}}}^*$ $(\bar{\imath} = 1, \ldots, d)$	$V_{(1+\bar{\imath})_\mathsf{r}}$	$=$	$v_\mathsf{r} T_\mathsf{r}^{-1}$
V_{2+d}	${V_{2+d}}^*$	$V_{(2+d)_\mathsf{r}}$	$=$	T_r^{-1}

Table 2.3: Dimensionless variables used together with the reference values they are based on. All reference values with an index r are based on the four quantities $L_\mathsf{r}, \rho_\mathsf{r}, v_\mathsf{r}, T_\mathsf{r}$. Reference values that introduce additional scales are given the index r'. To retain the distinction for the dimensionless quantities, they are denoted by a superscript * or $'$, depending on the type of their reference value. In the lowest part of the table, also the dimensionless form and reference values for entropy variables are listed.

of powers of the units according to the dimension of the formula. Where appropriate, the dimensionless numbers (2.48)–(2.52) are inserted.

2.10.1 Non-dimensionalization of the conservation equations

The non-dimensionalization of the conservation equations is straightforward; for example the energy equation (2.14c), multiplied by $L_\mathrm{r}/(\rho_\mathrm{r} v_\mathrm{r}^3)$ yields

$$
\begin{aligned}
&\frac{\partial}{\partial t^*}[\rho^*(e^* + k^*)] + \nabla^* \cdot [(\rho^*(e^* + k^*) + p^*)\boldsymbol{v}^*] \\
=&\nabla^* \cdot \Bigg[\Bigg\{\left(\lambda' \frac{\lambda_{\mathrm{r}'}}{\rho_\mathrm{r} v_\mathrm{r} L_\mathrm{r}} - \frac{2}{3}\eta' \frac{\eta_{\mathrm{r}'}}{\rho_\mathrm{r} v_\mathrm{r} L_\mathrm{r}}\right)(\nabla^* \cdot \boldsymbol{v}^*)\mathbb{I} \\
&\quad + \eta' \frac{\eta_{\mathrm{r}'}}{\rho_\mathrm{r} v_\mathrm{r} L_\mathrm{r}}(\nabla^* \boldsymbol{v}^* + (\nabla^* \boldsymbol{v}^*)^\mathsf{T})\Bigg\} \cdot \boldsymbol{v}^*\Bigg] + \nabla^* \cdot \left(\kappa' \frac{\kappa_{\mathrm{r}'} T_\mathrm{r}}{\rho_\mathrm{r} v_\mathrm{r}^3 L_\mathrm{r}} \nabla^* T^*\right) + \frac{f_\mathrm{r} L_\mathrm{r}}{v_\mathrm{r}^2}\boldsymbol{f}^* \cdot \boldsymbol{v}^* ,
\end{aligned}
$$

and by introducing the dimensionless numbers, the right hand side takes the form

$$
\begin{aligned}
=&\nabla^* \cdot \Bigg[\Bigg\{\left(\frac{\lambda'}{\mathrm{Re}_\lambda} - \frac{2}{3}\frac{\eta'}{\mathrm{Re}_\eta}\right)(\nabla^* \cdot \boldsymbol{v}^*)\mathbb{I} \\
&\quad + \frac{\eta'}{\mathrm{Re}_\eta}(\nabla^* \boldsymbol{v}^* + (\nabla^* \boldsymbol{v}^*)^\mathsf{T})\Bigg\} \cdot \boldsymbol{v}^*\Bigg] + \nabla^* \cdot \left(\frac{\kappa'}{\mathrm{Re}_\eta\ \mathrm{Ec}\ \mathrm{Pr}} \nabla^* T^*\right) + \frac{f_r L}{v_r^2}\boldsymbol{f}^* \cdot \boldsymbol{v}^* ,
\end{aligned}
\tag{2.53}
$$

which is identical to the original dimensional equation with all variables replaced by their dimensionless version and—where $(\cdot)'$-quantities have been used—the dimensionless numbers (2.48)–(2.52) premultiplied appropriately to the different terms.

2.10.2 Non-dimensionalization of the equations of state of specific fluid models

As an example of the non-dimensionalization of equations of state, those for the ideal gas will be treated here. An ideal gas is described by the equations of state (2.30a) and (2.30b). The state (p_0, ρ_0, T_0) is assumed to fulfill Eq. (2.30b) and for simplicity the (arbitrary) integration constant in the energy EOS is set to $e_0 = c_\mathrm{v} T_0$, so that $e = c_\mathrm{v} T$. The non-dimensionalization proceeds by dividing by the appropriate reference quantities as given by Table 2.3 to obtain

$$
e^* = \frac{1}{\gamma \mathrm{Ec}} T^* , \tag{2.54a}
$$

$$
p^* = \rho^* R^* T^* , \tag{2.54b}
$$

where $R^* = R/R_\mathrm{r} = (\gamma - 1)/(\gamma\,\mathrm{Ec})$ since $c_{\mathrm{p}_{\mathrm{r}'}} = c_\mathrm{p} = \text{const}$, and c_v accordingly.

For the transformation to other variable sets, one also needs an expression for the entropy, which is given by Eq. (2.31). To highlight a particular point in the non-dimensionalization, the cited relation for $s(\alpha, T)$ is replaced by $s(p, \rho)$, which is given by

$$s(p,\rho) = s_0 + \frac{R}{\gamma - 1} \log\left(\frac{p}{p_0}\right) - \frac{\gamma R}{\gamma - 1} \log\left(\frac{\rho}{\rho_0}\right) . \tag{2.55}$$

First of all, the constant s_0 is chosen such that $s(p_0, T_0) = 0$. As the base state quantities p_0 and ρ_0 still occur in (2.55), it is time to specify how the base state relates to the reference state. Two thermodynamic quantities are included in the set of reference quantities used for the non-dimensionalization, namely ρ_r and T_r. For simplicity we set $\rho_0 = \rho_\mathrm{r}$ and $T_0 = T_\mathrm{r}$. The equation of state (2.30b) has to be fulfilled for the base state, hence $p_0 = \rho_0 \, R \, T_0 = \rho_\mathrm{r} \, R_\mathrm{r} R^* \, T_\mathrm{r}$. On the other hand, the reference quantities have to fulfill $p_\mathrm{r} = \rho_\mathrm{r} \, R_\mathrm{r} \, T_\mathrm{r}$, so that the base pressure is $p_0 = p_\mathrm{r} \, R^*$. Consequently, the non-dimensionalized relation for entropy in terms of pressure and density is

$$s^* = \frac{R^*}{\gamma - 1} \left[\log\left(\frac{p^*}{R^*}\right) - \gamma \log \rho^* \right] = R^* \log \alpha^* + \frac{1}{\gamma \mathrm{Ec}} \log T^* . \tag{2.56}$$

The Eckert number needs to be related to the frequently provided Mach number. For an ideal gas this is achieved by inserting the expression for the velocity of sound from Eq. (2.33) into the definition of the Mach number, $M := v_\mathrm{r}/a$. This leads to

$$\mathrm{Ec} = \frac{v_\mathrm{r}^2}{c_{\mathrm{p_r}'} T_\mathrm{r}} = (\gamma - 1) M^2 . \tag{2.57}$$

The non-dimensionalization of the equations of state for other media proceeds analogously.

2.10.3 Non-dimensionalization of the quasi-linearized equations for entropy variables

To non-dimensionalize the quasi-linear equations, the dimensions of the entropy variable vector V, cf. Eq. (2.47), are considered and a dimensionless set V^* is defined. First, let

$$D_\mathrm{r} := \mathrm{diag}(v_\mathrm{r}, \, (v_\mathrm{r}^2)_{\bar{\imath}}, \, v_\mathrm{r}^3) , \tag{2.58}$$

where $(v_\mathrm{r}^2)_{\bar{\imath}}$ denotes d entries of v_r^2. The entropy variable vector can be factored in reference and dimensionless values as follows,

$$V = \frac{1}{T}\left(\mu - \frac{1}{2}v_n^2,\ v_{\bar{\imath}},\ -1\right)^{\mathsf{T}} \tag{2.59}$$

$$= \operatorname{diag}(v_{\mathrm{r}}^2 T_{\mathrm{r}}^{-1},\ (v_{\mathrm{r}} T_{\mathrm{r}}^{-1})_{\bar{\imath}},\ T_{\mathrm{r}}^{-1})\,\underbrace{\frac{1}{T^*}\left(\mu^* - \frac{1}{2}v_{\bar{n}}^{*2},\ v_{\bar{\imath}}^*,\ -1\right)^{\mathsf{T}}}_{V^*} \tag{2.60}$$

$$= \frac{v_{\mathrm{r}}^3}{T_{\mathrm{r}}} D_{\mathrm{r}}^{-1} V^*\,. \tag{2.61}$$

For the flux Jacobians the same procedure yields

$$A_{\bar{\imath}}^V = \frac{\rho_{\mathrm{r}} T_{\mathrm{r}}}{v_{\mathrm{r}}^3} D_{\mathrm{r}} A_{\bar{\imath}}^{V^*} D_{\mathrm{r}}\,, \quad \bar{\imath} = 1, \dots, d\,. \tag{2.62}$$

Remark 2.7 *The symmetric pre- and post-multiplication with the same diagonal matrix D_{r} ensures that the dimensionless flux Jacobian $A_i^{V^*}$ inherits the symmetry property of A_i^V. The same holds for the diffusive flux Jacobians, as long as they are symmetric. For example, the diffusive flux matrix K_{11}^V for V-variables for the spatial dimension $d = 3$ results from the dimensional version (cf. p. 144) as*

$$K_{11}^V = \frac{\rho_r L_r T_r}{v_r^3} D_r\, T^* \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \frac{\lambda'}{\mathrm{Re}_\lambda} + \frac{4}{3}\frac{\eta'}{\mathrm{Re}_\eta} & 0 & 0 & \left(\frac{\lambda'}{\mathrm{Re}_\lambda} + \frac{4}{3}\frac{\eta'}{\mathrm{Re}_\eta}\right) v_1^* \\ & & \frac{\eta'}{\mathrm{Re}_\eta} & 0 & \frac{\eta'}{\mathrm{Re}_\eta} v_2^* \\ & & & \frac{\eta'}{\mathrm{Re}_\eta} & \frac{\eta'}{\mathrm{Re}_\eta} v_3^* \\ & \text{sym.} & & & \frac{2\eta'}{\mathrm{Re}_\eta} k^* + \left(\frac{\lambda'}{\mathrm{Re}_\lambda} + \frac{1}{3}\frac{\eta'}{\mathrm{Re}_\eta}\right) v_1^{*2} + \frac{\kappa' T^*}{\mathrm{Re}_\eta \mathrm{Ec}\,\mathrm{Pr}} \end{pmatrix}}_{=:\ K_{11}^{V\,*}} D_r\,.$$

□

For the terms of the quasi-linear equation that means

$$A_{\bar{n}}^V V_{,\bar{n}} = \frac{T_{\mathrm{r}}\rho_{\mathrm{r}}}{v_{\mathrm{r}}^3} D_{\mathrm{r}}\, A_{\bar{n}}^{V^*} D_{\mathrm{r}} \frac{v_{\mathrm{r}}^3}{T_{\mathrm{r}}} D_{\mathrm{r}}^{-1} \frac{1}{L_{\mathrm{r}}} V^*_{,x^*_{\bar{n}}} \tag{2.63}$$

$$= \frac{\rho_{\mathrm{r}}}{L_{\mathrm{r}}} D_{\mathrm{r}}\, A_{\bar{n}}^{V^*} V^*_{,x^*_{\bar{n}}} \tag{2.64}$$

$$(K_{\bar{m}\bar{n}}^V V_{,\bar{n}})_{,\bar{m}} = \left(\frac{\rho_{\mathrm{r}} L_{\mathrm{r}} T_{\mathrm{r}}}{v_{\mathrm{r}}^3} D_{\mathrm{r}} K_{\bar{m}\bar{n}}^{V\,*} D_{\mathrm{r}} \frac{v_{\mathrm{r}}^3}{T_{\mathrm{r}}} D_{\mathrm{r}}^{-1} \frac{1}{L_{\mathrm{r}}} V^*_{,x^*_{\bar{n}}}\right)_{,x^*_{\bar{n}}} \frac{1}{L_{\mathrm{r}}} \tag{2.65}$$

$$= \frac{\rho_{\mathrm{r}}}{L_{\mathrm{r}}} D_{\mathrm{r}} \left(K_{\bar{m}\bar{n}}^{V\,*} V^*_{,x^*_{\bar{n}}}\right)_{,x^*_{\bar{m}}}\,. \tag{2.66}$$

Multiplication with the inverse of the common factor $(\rho_\mathrm{r}/L_\mathrm{r})\,D_\mathrm{r}$ recovers the same functional form as in Eq. (2.43). This result coincides with the one of Eq. (2.53), namely that the dimensionless prognostic equations are identical to the dimensional ones except for the scaling of different terms by (products of) the dimensionless numbers (2.48)–(2.51).[14] Therefore, in the sequel no explicit distinction will be made between the equations and variables in dimensional and dimensionless form.

2.11 Conclusions

The behavior of fluids has been modeled through application of physical conservation statements for the macroscopic dynamic motion, and with equations of state for their reaction to changes of the thermodynamical state. Through non-dimensionalization, extraneous standards have been removed. All equations keep their functional form, usually with weighting factors in the form of dimensionless constants for different terms. Thanks to the coincident functional dependence, no difference needs to be made between the dimensional and dimensionless forms. In the sequel, the non-dimensional equations are standard, but the explicit notation with superscripts as in q^* and q' is omitted.

The mathematical model has been extended by generalizing the set of independent variables. Different choices for the set of variables have been cited and special properties emphasized. In particular, the applicability to both compressible and incompressible flows provides a promising starting point regarding the first central question formulated in the introduction. The generalized variable approach—and in particular entropy variables—will be exploited in the sequel. The goal is to construct numerical methods that retain the property of the fundamental equations and are applicable to fluids of different nature. This will be the topic of Chapters 4 and 5. First, however, common definitions are given in the following chapter.

[14] In short, the following replacements take place in the conservation equations: $\lambda \to \lambda'/\mathrm{Re}_\lambda$, $\eta \to \eta'/\mathrm{Re}_\eta$, $\kappa \to \kappa'/(\mathrm{Re}_\eta \mathrm{Ec}\,\mathrm{Pr})$.

Chapter 3

Mathematical formalization

Vaka vanha Väinämöinen näki tyrskyn työntelevän,
hyrskyn maalle hylkeävän, aallon rannallen ajavan
noita sampuen muruja, kirjokannen kappaleita.

Hän tuosta toki ihastui. Sanan virkkoi, noin nimesi:
"Tuost' on siemenen sikiö, alku onnen ainiaisen,
tuosta kyntö, tuosta kylvö, tuosta kasvu kaikenlainen!"

Elias Lönnrot (1802–1884), *Kalevala*

3.1 Introduction

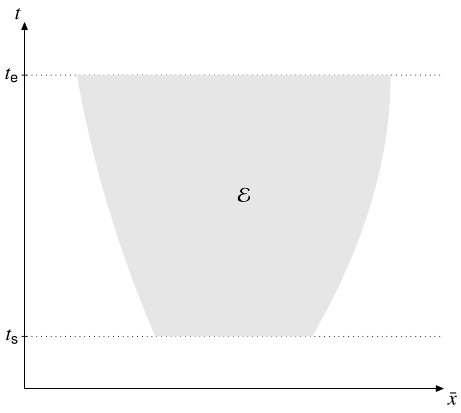
This chapter lays the foundation for the discussion of the least-squares and discontinuous Galerkin finite element discretizations in Chapters 4 and 5, respectively. General definitions for different mesh entities are given here and the corresponding notation is established.

3.2 Space-time geometry

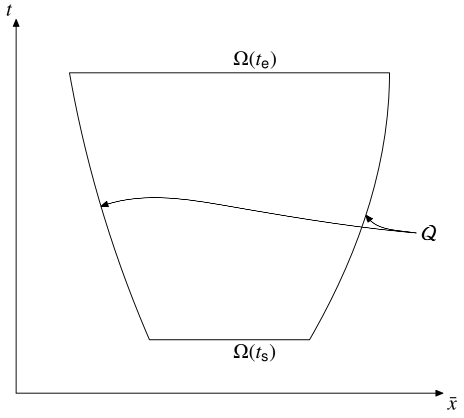
A flow problem is considered in space and time, starting at time t_s and ending at t_e. The spatial domain of the flow problem may be time-dependent. The time dependence can either be prescribed, e.g. moving solid boundaries, or itself be part of the solution process, as in free-surface problems.

Let the physical space dimension of the problem be d. Adding the time-dependence, $\mathcal{E} \subset \mathbb{R}^{1+d}$ is the open *space-time domain* of the problem. A point in the domain is given by its position vector $\boldsymbol{x}$. Whenever the vector character is irrelevant, the point is denoted x and, depending on the context, it is represented by its coordinates with respect to a Cartesian coordinate system as $x = (x_0, x_1, \ldots, x_d) = (x_0, \bar{x}) = (t, \bar{x})$. As introduced in Chapter 2, the overbar indicates extent over spatial dimensions, like the spatial coordinates $\bar{x} = (x_1, \ldots, x_d)$, as opposed to the space-time position x.

The spatial flow domain at time t is $\Omega(t) := \{x \in \mathbb{R}^{1+d} \mid (t, \bar{x}) \in \mathcal{E}\}$. The boundary of the space-time domain, $\partial\mathcal{E}$, can be decomposed into the spatial domain at the start and end time, $\Omega(t_\mathsf{s})$ and $\Omega(t_\mathsf{e})$, respectively, which are both subsets of hyperplanes in $\mathbb{R}^{1+d}$, and the hypersurface spanned by the spatial domain boundary $\partial\Omega(t)$ and time, $\mathcal{Q} := \{x \in \partial\mathcal{E} \mid t_\mathsf{s} < x_0 < t_\mathsf{e}\}$. See Figure 3.1 for examples with the spatial dimension $d = 1$.



(a) The time-dependent line segment $\Omega(t) \subset \mathbb{R}$ for $t \in [t_s; t_e]$ defines the space-time domain $\mathcal{E} \subset \mathbb{R}^2$.



(b) The boundary $\partial\mathcal{E}$ of the space-time domain.

Figure 3.1: The space-time domain $\mathcal{E}$ and associated notation.

Domain discretization

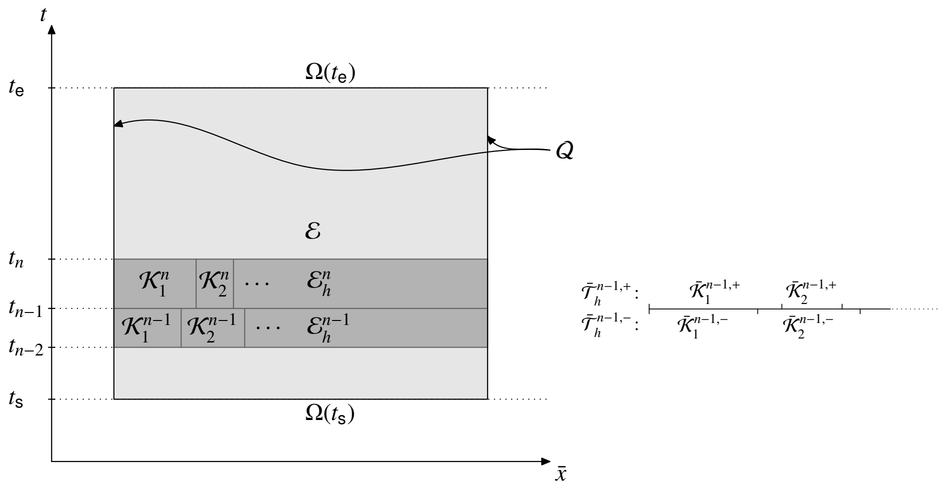
To make the space-time domain $\mathcal{E}$ available to finite element methods, it has to be partitioned by a suitable mesh. First, the considered time interval $\mathtt{T} := [t_s; t_e]$ is subdivided into N_t intervals $\mathtt{T}^n := [t_{n-1}; t_n]$, with $t_s = t_0 < t_1 < \ldots < t_{N_t} = t_e$. The space-time domain $\mathcal{E}$ is divided by the hyperplanes $\mathcal{H}^n := \{x \in \mathbb{R}^{1+d} \mid x_0 = t_n\}$ into *space-time slabs* $\mathcal{E}^n := \{x \in \mathcal{E} \mid x_0 \in \mathtt{T}^n\}$.

Starting with the space domain $\Omega(t_{n-1})$ at time t_{n-1}, its evolution during $\mathtt{T}^n$ is given by a time-dependent mapping

$$\begin{aligned} \Phi_t^n \colon \Omega(t_{n-1}) &\to \Omega(t), \quad t \in \mathtt{T}^n\,, \\ \bar{x} &\mapsto \Phi_t^n(\bar{x})\,, \end{aligned}$$

which is assumed to be sufficiently smooth, orientation preserving and invertible. At each time t_{n-1}, a tessellation $\bar{\mathcal{T}}^{n-1,+}$ of the physical space domain $\Omega(t_{n-1})$ into N_{el}^n elements $\bar{\mathcal{K}}_e^{n-1,+}$, $e = 1, \ldots, N_{\mathrm{el}}^n$, is given. Each of these elements is related to a suitable reference element $\hat{\bar{\mathcal{K}}}_e^{n-1}$ by the mapping $\bar{G}_e^{n-1,+} \colon \hat{\bar{\mathcal{K}}}_e^{n-1} \to \bar{\mathcal{K}}_e^{n-1,+}$. The reference shapes used for the considered dimensions d are listed in Table 3.1 on the next page.

By applying the mapping $\Phi_{t_n}^n$, the image of each element $\bar{\mathcal{K}}_e^{n-1,+}$ at time t_n is found; the image is denoted $\bar{\mathcal{K}}_e^{n,-}$, and the set of all images of the elements in $\bar{\mathcal{T}}^{n-1,+}$ defines the tessellation $\bar{\mathcal{T}}^{n,-}$, cf. Figure 3.2b on the facing page. The space-time elements $\mathcal{K}_e^n$ in slab $\mathcal{E}^n$ are defined by linear interpolation of $\bar{\mathcal{K}}_e^{n-1,+}$ and $\bar{\mathcal{K}}_e^{n,-}$ in time, cf. Figure 3.2a. Finally, the set of all space-time elements in the nth slab constitutes the


t
t_e
t_n
t_{n-1}
t_{n-2}
t_s
$\Omega(t_e)$
Q
$\mathcal{E}$
$\mathcal{K}_1^n$ $\mathcal{K}_2^n$ $\cdots$ $\mathcal{E}_h^n$
$\mathcal{K}_1^{n-1}$ $\mathcal{K}_2^{n-1}$ $\cdots$ $\mathcal{E}_h^{n-1}$
$\Omega(t_s)$
$\bar{x}$
$\bar{\mathcal{T}}_h^{n-1,+}$:
$\bar{\mathcal{T}}_h^{n-1,-}$:
$\bar{\mathcal{K}}_1^{n-1,+}$
$\bar{\mathcal{K}}_2^{n-1,+}$
$\bar{\mathcal{K}}_1^{n-1,-}$
$\bar{\mathcal{K}}_2^{n-1,-}$

(a) Space-time domain and mesh.

(b) Space-tessellations at an inter-time slab boundary.

Figure 3.2: Illustration of the notation for a mesh on a one-dimensional space domain (here without mesh deformation and moving boundaries).

space-time tessellation $\mathcal{T}^n = \{\mathcal{K}_e^n \mid e = 1 \dots N_{\mathsf{el}}^n\}$, and the domain of the discretized time slab is $\mathcal{E}_h^n := \bigcup_{e=1}^{N_{\mathsf{el}}^n} \mathcal{K}_e^n$, with h a representative measure of the minimal cell diameter. The discrete space-time domain boundary $\mathcal{Q}_h$ is defined analogously to $\mathcal{Q}$ and its restriction to the n^{th} slab is $\mathcal{Q}_h^n$.

Remark 3.1 *In the above framework, two different space tessellations may exist on the sides of each hyperplane $\mathcal{H}^n$, allowing to re-mesh in case the physical space tessellation does not fulfill the quality requirements after applying the mapping $\Phi_{t_n}^n$.* □

Like the space reference element $\hat{\bar{\mathcal{K}}}_e^{n-1}$ for each element $\bar{\mathcal{K}}_e^{n-1,+}$, the space-time element $\mathcal{K}_e^n$ has a space-time reference element $\hat{\mathcal{K}}_e^n$. The space-time reference elements are defined by extending the reference shapes of Table 3.1 with a time dimension: $\hat{\mathcal{K}}_e^n = \hat{\mathrm{T}} \times \hat{\bar{\mathcal{K}}}_e^{n-1}$ with

Table 3.1: The reference shapes used for the discretization in physical space with dimension d; see also Chapter 6 and Appendix B.

d	shape	# of faces	# of nodes
1	line	2	–
2	triangle	3	3
	square	4	4
3	tetrahedron	4	4
	pyramid	5	5
	triangular prism	5	6
	cube	6	8

the reference time interval $\hat{\mathrm{T}} := [-1; 1]$. The mapping from reference time $\hat{x}_0$ to physical time t is defined as

$$\begin{aligned} G_{\hat{\mathrm{T}}}^{\mathrm{T}^n} \,:\, \hat{\mathrm{T}} &\to [t_{n-1}; t_n]\,, \\ \hat{x}_0 &\mapsto t = \tfrac{t_n + t_{n-1}}{2} + \tfrac{t_n - t_{n-1}}{2} \hat{x}_0\,. \end{aligned}$$

This also exemplifies the general notation G_A^B for mappings from A to B. For the mapping from the space-time reference element to physical space-time element,

$$\begin{aligned} G_{\hat{\mathcal{K}}_e^n}^{\mathcal{K}_e^n} : \hat{\mathcal{K}}_e^n &\to \mathcal{K}_e^n\,, \\ \hat{x} &\mapsto \left(G_{\hat{\mathrm{T}}}^{\mathrm{T}^n}, \tfrac{1}{2}(1 - \hat{x}_0)\, \bar{G}_e^{n-1,+}(\hat{\bar{x}}) + \tfrac{1}{2}(1 + \hat{x}_0)\, \bar{G}_e^{n,-}(\hat{\bar{x}}) \right)\,, \end{aligned}$$

the abbreviation G_e^n is frequently used. Excluding the time coordinate from the element mapping yields the (time-dependent) space mapping $\bar{G}_e^n$.

Remark 3.2 *The mapping between reference time $\hat{x}_0$ and physical time x_0 is independent of space but the mapping of the space coordinates is time-dependent (in deforming meshes).* □

Remark 3.3 *It is also possible to consider subtimestepping; hereby the time interval is split by some elements $\mathcal{K}_e^n$ in such a way that at least one of the intersections $\mathcal{K}_e^n \cap \mathcal{H}^{n-1}$ and $\mathcal{K}_e^n \cap \mathcal{H}^n$ is empty. This case could be covered by the element enumeration per slab rather than by the slab superscript n. However, this technique is not used here. Therefore the mapping from reference to physical time, $G_{\hat{\mathrm{T}}}^{\mathrm{T}^n}$, is the same for all elements, thus simplifying the notation.* □

Remark 3.4 *Elements may be added or removed from the space discretization within one time interval by setting the space-volume at the start or end of the time slab to zero. To realize this, moving element boundaries and an arbitrary Lagrangian–Eulerian formulation are necessary (van der Vegt and van der Ven, 2002b). The element mapping Φ_t^n is not invertible at the concerned time level, but in practice that is immaterial.* □

3.3 Faces

Each space reference element $\hat{\bar{\mathcal{K}}}_e^n$ listed in Table 3.1 is characterized by its geometry as given by the number of nodes, their positions and the connectivity in objects of codimension greater or equal to one, see Appendix B. In the following, the entities of codimension one (with regard to the total space or space-time dimension) are called the element faces $\bar{F}_{e,j}^{n-1,+}$ of the space element $\bar{\mathcal{K}}_e^{n-1,+}$. The element face $\bar{F}_{e,j}^{n-1,+}$ is extended in the same way into space-time as the elements: based on its image under the mapping $\Phi_{t_n}^n$, the linear interpolation between the two time levels defines $F^n_{e,j}$. The space-time element $\mathcal{K}^n_e$ is then bounded by

$$\partial\mathcal{K}^n_e = \bar{\mathcal{K}}^{n-1,+}_e \cup \bar{\mathcal{K}}^{n,-}_e \cup \bigcup_j F^n_{e,j} .$$

For the discontinuous Galerkin method developed in Chapter 5 it is beneficial to see a face not as induced by an element but as independent constituent of the mesh. The following definition makes that possible and distinguishes several kinds of faces based on their function in the space-time slab $\mathcal{E}^n$.

Definition 3.1 (Faces): *The element face $F^n_{e,j}$ is tagged as*

- *an internal face $\mathcal{S}^{\mathsf{i},n}_{\{e,e'\}}$, if $\exists\, e, e' \in \{1, \ldots, N^n_{\mathsf{el}}\},\ e \neq e',\ j, j' : F^n_{e,j} = F^n_{e',j'}$,*
- *a* (past) time face $\mathcal{S}^{\mathsf{p},n}_{e,e'}$, *if $\exists\, e \in \{1, \ldots, N^n_{\mathsf{el}}\},\ e' \in \{1, \ldots, N^{n-1}_{\mathsf{el}}\},\ j, j' : F^n_{e,j} = F^{n-1}_{e',j'}$,*
- *a* (future) time face $\mathcal{S}^{\mathsf{f},n}_{e,e'}$, *if $\exists\, e \in \{1, \ldots, N^n_{\mathsf{el}}\}, e' \in \{1, \ldots, N^{n+1}_{\mathsf{el}}\}, j, j' : F^n_{e,j} = F^{n+1}_{e',j'}$,*
- *a boundary face $\mathcal{S}^{\mathsf{b},n}_{e,j}$ otherwise.*

All boundary faces of the space-time slab $\mathcal{E}^n_h$ are subsumed in the set $\mathcal{F}^{\mathsf{b},n}$ and the internal faces in the set $\mathcal{F}^{\mathsf{i},n}$. Finally, all faces of these two types are included in the set $\mathcal{F}^n := \mathcal{F}^{\mathsf{i},n} \cup \mathcal{F}^{\mathsf{b},n}$. The past and future time faces of the slab are included in $\mathcal{F}^{\mathsf{p},n}$ and $\mathcal{F}^{\mathsf{f},n}$, respectively, and $\mathcal{F}^{\mathsf{t},n} := \mathcal{F}^{\mathsf{p},n} \cup \mathcal{F}^{\mathsf{f},n}$. □

Remark 3.5

1. *According to the previous definitions, the space-time element faces in the hyperplanes $\mathcal{H}^0$ and $\mathcal{H}^{N_t}$, i.e., at the start and end time, respectively, are boundary faces. But for practical matters they do not require special treatment, rather the same (upwind) flux is applied as on time faces, cf. Section 5.2.5.*
2. *The set notation for the subscript of an internal face $\mathcal{S}^{\mathsf{i},n}_{\{e,e'\}}$ indicates the uniqueness of the face, thus an interchange of the two indices refers to the same entity.*
3. *For the definition of the faces it is assumed that all elements are convex, which can always be guaranteed by subdivision.* □

Definition 3.2 (Traces): *For a function w defined on the space-time domain $\mathcal{E}$, the traces in a point $\boldsymbol{x}$ at the boundary $\partial\mathcal{K}^n_e$ of the element $\mathcal{K}^n_e$ are defined as*

$$w^{\pm}_{\mathcal{K}^n_e} := \lim_{\epsilon\downarrow 0} w(\boldsymbol{x} \pm \epsilon \boldsymbol{n}_{\mathcal{K}^n_e}) .$$

As the normal vector $\boldsymbol{n}_{\mathcal{K}^n_e}$ is outward with respect to the element $\mathcal{K}^n_e$, the superscript '+' defines the exterior trace and '−' the interior trace. If $\boldsymbol{x}$ is located on the boundary $\partial\mathcal{E}$ of the space-time domain $\mathcal{E}$ then only the interior trace $w^-_{\mathcal{K}^n_e}$ exists. □

3.4 Notational conventions

The summation convention is implied on repeated indices; where possible, summation indices use the letters $m, \dots, s$. The letter n frequently does not indicate a summation but rather serves as the time slab index, or, occasionally, denotes the components n_i of a normal vector $\boldsymbol{n}$. The indices i, j, k are usually free indices, chosen within a suitable range, typically the space or space-time dimension. All dimensional indices come in two versions: unaccented (i) or with overbar ($\bar{\imath}$). While the unaccented version concerns the time and space dimensions, $i = 0, \dots, d$, the overbar version covers space dimensions only, $\bar{\imath} = 1, \dots, d$, see also Section 2.5. Indices in $\mathsf{sans\text{-}serif}$ font are textual rather than mathematical entities. Mathematical subscripts with comma notation indicate partial differentiation, e.g. $U_{,t} = \partial U/\partial t$, $U_{,i} = \partial U/\partial x_i$. When indexing components of a state data vector, e.g. U or V, the index is marked with a tilde, i.e., $\tilde{\imath} = 1, \dots, 2 + d$.

Chapter 4

A stabilized Galerkin least-squares finite element method for the Navier–Stokes equations

Le temps use l'erreur et polit la verité.

François Gaston de Lévis (1719–1787),
Maximes, préceptes et réflexions

4.1 Introduction

Finite element methods furnish a procedure to compute approximations of the solutions of (partial) differential equations. The Galerkin finite element procedure—derive a weak form for a given PDE, discretize the function spaces with sufficiently smooth basis functions and solve the resulting finite-dimensional problem—was first successfully applied to elliptic equations. It soon became apparent that using the method for other types of PDEs leads to problems. The first of these problems is related to advection-dominated flow, for which the Galerkin FEM lacks stability and spurious oscillations can degrade the solution, see, e.g., (Franca et al., 1992). Various methods with an 'upwind' component have been developed to overcome the stability problem. In particular the Petrov–Galerkin method, which uses different basis functions for the test and trial space, has been used to stabilize finite element discretizations. More recently, the streamline upwind Petrov–Galerkin (SUPG) method has been devised. It introduces stabilization only in the streamline direction, which improves the accuracy of the method.

Avoiding upwinding and artificial diffusion-related ideas, the Galerkin least-squares method constitutes a different approach to obtain a stable numerical method. By adding a least-squares term to the standard Galerkin formulation this method solves the stability problem without reducing the order of accuracy of the method, provided a suitably defined least-squares operator is used. Interestingly, the least-squares term stabilizes the Galerkin method not only with respect to the advective effect. It also defeats the problem of pressure-field oscillations which occurs in the treatment of the incompressible Navier–Stokes equations. That means that the Ladyzhenskaya–Babuška–Brezzi (LBB) condition (also called the inf-sup-condition) is satisfied by the Galerkin least-squares ansatz without resorting to *mixed methods*, i.e., choosing different approximation spaces for the different solution components.

When based on a suitable formulation of the Navier–Stokes equations, the Galerkin least-squares FEM is applicable to both compressible and incompressible flow. Therefore, the generalized variable approach (cf. Chapter 2)—in which the conservative base set may be replaced by other sets, like entropy variables—has been at the heart of a number of finite element discretizations for the Euler and Navier–Stokes equations (Hughes et al., 1986; Shakib et al., 1991; Barth, 1999). An important observation is that for entropy variables the Navier–Stokes equations have a well-defined incompressible limit. Provided all components of the numerical method maintain this property, stable and efficient algorithms can be designed that apply to both compressible and incompressible flow. This route was taken by Hauke and Hughes (1998), who give numerical examples for both cases computed with a Galerkin least-squares FEM. The line of thought followed here aims to support the general applicability of the entropy variable formulation by designing a single stabilization operator that applies to both kinds of flow. For that matter, the results presented in this chapter are a continuation of the research of Polner et al. (2006). To complete their work, which focussed on the application of a newly designed stabilization operator to incompressible flows using pressure primitive variables, Polner, Pesch, and van der Vegt (2007) have generalized the definition of the stabilization operator and implemented the method successfully for entropy variables and compressible fluids. The results demonstrate that the new stabilization operator, whose definition includes the compressibility properties of the fluid, applies to both compressible and incompressible flows and does not negatively affect the accuracy of the finite element discretization.

This chapter presents a continuous finite element formulation for the Navier–Stokes equations using entropy variables (Hauke and Hughes, 1998), the new stabilization operator in the Galerkin least-squares formulation, and numerical results for compressible flow. The latter are contributions to (Polner et al., 2007), where the theoretical derivation of the stabilization operator can be found. In Section 4.2 the function spaces for the FE discretization are identified. For the purpose of this chapter, these spaces are subsets of the Sobolev space $H^1(\mathcal{E}_h^n)$, which requires that functions and their weak derivative are square integrable. Section 4.2 also contains the weak form of the Navier–Stokes equations. The evaluation of the integrals that occur in the weak form is detailed, as is their linearization, which is necessary for the solution of the nonlinear system of equations with a Newton method. Details about the (numerical) boundary conditions are discussed in Section 4.3, before the main subject of (Polner et al., 2007), the stabilization matrix for compressible and incompressible fluids, is stated in Section 4.4. Section 4.5 presents the solutions for several test problems as evidence that the stabilization operator fulfills the requirements. Section 4.6 contains an evaluation of the numerical method presented in this chapter and conclusions.

4.2 Weak formulation and discretization

The starting point for the finite element discretization of the PDE system at hand is the weak formulation, which is obtained by multiplying the equations with arbitrary test functions and integrating over the domain. This step is, however, only meaningful when the function spaces containing the trial and test functions are properly specified. While in this chapter the space discretization uses continuous basis functions, the discretization in time is discontinuous, using either constant or linear-in-time functions. The discontinuous basis functions in time allow remeshing between timesteps, so that—together with the natural arbitrary Lagrangian/Eulerian (ALE) formulation of the equations—moving meshes can be treated by the method. This possibility is, however, not exploited by the implementation and test cases presented here, hence the ALE form of the equations, which can be found in (Polner, 2005), is not derived. The time discretization is implicit through the (approximate) integration over the space-time slabs.

4.2.1 Finite element spaces

The function spaces used to discretize a weak formulation necessarily have to be finite-dimensional to allow a numerical treatment. Finite-dimensionality is achieved by allowing only functions that are piecewise—i.e. per element in the tessellation $\mathcal{T}$—of a certain polynomial degree. In this chapter, the trial and test function spaces for the space-time slab $\mathcal{E}_h^n$, $\mathcal{W}_h^{n,(p_\text{t},p_\text{s})}$ and $\mathcal{Y}_h^{n,(p_\text{t},p_\text{s})}$, respectively, are defined based on the tessellation of the n^th slab, $\mathcal{T}^n$, so that they contain globally C^0-continuous functions, which are elements of the space of tensor-product polynomials $P_{(p_\text{t},p_\text{s})}(\hat{\mathcal{K}}_e^n)$ of degree p_t in time and p_s in space on the reference element of each element $\mathcal{K}_e^n$. The spatial degree is fixed to $p_\text{s} = 1$ for the numerical examples given in this chapter. This restriction is, however, not exploited in the derivation of the discretization, so that higher order basis functions may be used with the same formulae. Further, all expansions are either constant or linear in time, viz. $p_\text{t} \in \{0, 1\}$. For the constant in time approximation, simplifications regarding the time integral can be applied, as will be pointed out in due course. The trial and test function spaces are given by

$$\begin{aligned}\mathcal{W}_h^{n,(p_\text{t},p_\text{s})} := \{V \in (H^1(\mathcal{E}_h^n))^{2+d} \mid V|_{\mathcal{K}_e^n} \circ G_e^n \in (P_{(p_\text{t},p_\text{s})}(\hat{\mathcal{K}}_e^n))^{2+d}\ \forall \mathcal{K}_e^n \in \mathcal{T}^n;\\ B_1(V) = 0 \text{ on } \mathcal{Q}_h^n\}\,,\end{aligned} \tag{4.1}$$

$$\begin{aligned}\mathcal{Y}_h^{n,(p_\text{t},p_\text{s})} := \{W \in (H^1(\mathcal{E}_h^n))^{2+d} \mid W|_{\mathcal{K}_e^n} \circ G_e^n \in (P_{(p_\text{t},p_\text{s})}(\hat{\mathcal{K}}_e^n))^{2+d}\ \forall \mathcal{K}_e^n \in \mathcal{T}^n;\\ B_2(W) = 0 \text{ on } \mathcal{Q}_h^n\}\,.\end{aligned} \tag{4.2}$$

The boundary operators B_1 and B_2 depend on the type and value of the boundary conditions on $\mathcal{Q}_h^n$, which have to be chosen appropriately for the given problem and set of variables, cf. Sections 4.3 and 4.5.

It should be noted that although the set of unknowns is called V here, all derivations in this chapter remain the same if a different set of variables, e.g. pressure primitive variables Y, is used.

4.2.2 Weak formulation

Integrating the set of conservation equations (2.14) over the discretized space-time slab $\mathcal{E}_h^n$ and applying the Gauß theorem results in flux terms on the space and time boundary of the slab. Adding a least-squares term for stability, the weak form is stated as:
Find $V \in \mathcal{W}_h^{n,(p_\mathrm{t},p_\mathrm{s})}$ such that for all $W \in \mathcal{Y}_h^{n,(p_\mathrm{t},p_\mathrm{s})}$ holds

$$\begin{aligned}\int\limits_{\mathcal{E}_h^n} \left(-W_{,r} \cdot F_r^{\mathsf{e}}(V) + W_{,\bar{r}} \cdot (K_{\bar{r}\bar{s}}^V V_{,\bar{s}})\right) \mathrm{d}\mathcal{E} + B_{\mathsf{ls}}(W,V) + B_{\mathsf{bc}}(W,V) \\ + \int\limits_{\Omega(t_n)} W(t_n^-) \cdot F_0^{\mathsf{e}}(V(t_n^-))\,\mathrm{d}\Omega - \int\limits_{\Omega(t_{n-1})} W(t_{n-1}^+) \cdot F_0^{\mathsf{e}}(V(t_{n-1}^-))\,\mathrm{d}\Omega = 0\,.\end{aligned} \tag{4.3}$$

The terms in the bracketed space-time slab integral in Eq. (4.3) are the Euler and diffusive fluxes. The original (i.e., not the quasi-linear) form of the equations is used, except in the least-squares term B_{ls}, which will be specified later. The boundary conditions are subsumed in the operator B_{bc}, and weak continuity in time is enforced by the integrals over the time boundaries with the previous and future time slab, $\Omega(t_{n-1})$ and $\Omega(t_n)$, respectively.

4.2.3 Construction of the basis functions

The test functions W and the trial functions V are discretely expanded using (i) continuous nodal basis functions in space, $\psi^{n,g} \colon \Omega(t) \to \mathbb{R}$, and (ii) discontinuous basis functions in time, $\lambda^{n,j} \colon [t_\mathsf{s}; t_\mathsf{e}] \to \mathbb{R}$:

(i) The spatial basis functions are associated with the nodes of the space tessellation $\bar{\mathcal{T}}^n$, that is, $\psi^{n,g}$ attains the value one at the node with index g and zero in all other nodes. In this way the support is minimal under the requirement of global continuity.
On the other hand, for computational reasons, an element-based view is preferable and the global basis functions are expressed in terms of functions $\hat{\psi}_i$ defined on a reference element. The local basis functions are defined in terms of the reference space coordinates $\hat{x}_i$ on the reference element $\hat{\mathcal{K}}_e^n$ and transformed to the physical space element $\mathcal{K}_e^n$ by the mapping G_e^n. The described requirements on the global basis functions correspond to Lagrange elements, see, e.g., (Brenner and Scott, 2002). Each global basis function can be represented as a sum of local basis functions from different elements: The global basis function $\psi^{n,g}$ is the sum of the local basis functions whose support is one of the elements that contain the node g and which are nonzero in g.

(ii) Throughout this chapter, the expansion in reference time $\hat{x}_0 \in \hat{\mathrm{T}}$ is either constant or linear, hence the number of time basis functions per time slab is $n_\mathrm{t} = 1$ or $n_\mathrm{t} = 2$, respectively, for all elements. The reference time basis function is $\hat{\lambda}^1(\hat{x}_0) = 1$ for the constant in time case and $\hat{\lambda}^1(\hat{x}_0) = \frac{1}{2}(1 - \hat{x}_0)$ and $\hat{\lambda}^2(\hat{x}_0) = \frac{1}{2}(1 + \hat{x}_0)$ for the linear in time case, so that the time basis functions in the actual space-time slab are given by

$$\lambda^{n,1} = \begin{cases} 1 & \text{if } t_{n-1} < t < t_n\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{4.4}$$

for constant in time, and

$$\lambda^{n,1} = \begin{cases} \frac{t_n - t}{t_n - t_{n-1}} & \text{if } t_{n-1} < t < t_n\,, \\ 0 & \text{otherwise}\,, \end{cases} \qquad \lambda^{n,2}(t) = \begin{cases} \frac{t - t_{n-1}}{t_n - t_{n-1}} & \text{if } t_{n-1} < t < t_n\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{4.5}$$

for linear in time.

Expansions in terms of element-local basis functions

The degrees of freedom of the unknown V are the expansion coefficients of the solution with respect to the chosen basis for the discrete function space. On a space-time reference element $\hat{\mathcal{K}}_e^n$, the variable V is expanded as

$$V_e^n(\hat{x}) = \sum_{i=1}^{n_\mathrm{s}} \sum_{j=1}^{n_\mathrm{t}} V_{i,e}^{n,j}\, \hat{\lambda}^j(\hat{x}_0)\, \hat{\psi}_i(\hat{\bar{x}})\,. \tag{4.6}$$

with the coefficient vectors $V_{i,e}^{n,j} \in \mathbb{R}^{2+d}$. However, not all the local expansion coefficients are independent since the actual degrees of freedom are the coefficients with respect to the underlying global basis functions; hence the assembly procedure has to map from local degrees of freedom to global ones, which are denoted $V_g^{n,j}$. Like n_t in the temporal expansion, the number of spatial basis functions, n_s, is assumed fixed for all elements.

Just as the original equations are nonlinear, so is the discrete system with respect to the degrees of freedom. Therefore, a linearization is carried out to apply a Newton-like solution strategy to the discrete problem. For the linearization of the global system arising from the discretization of the weak form (4.3) several differential relations involving element expansions are necessary. The differentiation of (4.6) with respect to an expansion coefficient $V_{i',e'}^{n',j'}$ yields

$$\frac{\partial V_e^n}{\partial V_{i',e'}^{n',j'}} = \delta_{ee'} \delta_{nn'} \hat{\lambda}^{j'} \hat{\psi}_{i'}\,, \qquad i' \in \{1, \dots, n_\mathrm{s}\},\ j' \in \{1, \dots, n_\mathrm{t}\},\ e, e' \in \{1, \dots, N_\mathrm{el}^n\}\,. \tag{4.7}$$

Differentiation of (4.6) with respect to the reference time yields

$$\frac{\partial V_e^n}{\partial \hat{x}_0} = \sum_{i=1}^{n_\mathsf{s}} \sum_{j=1}^{n_\mathsf{t}} V_{i,e}^{n,j} \frac{\partial \hat{\lambda}^j}{\partial \hat{x}_0} \hat{\psi}_i \,, \tag{4.8}$$

and the spatial derivative is similar except that the differentiation acts only on $\hat{\psi}_i$. These relations will be used in the next section where the integrals of the weak form are evaluated for trial and test functions from their respective discrete spaces, $\mathcal{W}_h^{n,(p_\mathsf{t},p_\mathsf{s})}$ and $\mathcal{Y}_h^{n,(p_\mathsf{t},p_\mathsf{s})}$.

4.2.4 Evaluation of weak form integrals, time integration, linearization

In this section, the integrals in the weak form (4.3) are evaluated further, based on the expansion (4.6) of the trial function, but still without specifying the test function. In the following, the symbol $V_e^n(\zeta)$ abbreviates $V_e^n(\hat{x}_0 = \zeta, \hat{\bar{x}})$ as a function of the reference space position at the given reference time $\hat{x}_0 = \zeta$. The evaluation of the integral over the time coordinate is done with the trapezoidal rule (TR), which yields a second order accurate approximation by averaging the values at the start and end of the time slab. If the expansion in time is merely constant, then a single evaluation of the integrand suffices and halves the computational cost of the integration. This case will not be considered separately below. Integration on space elements and faces is carried out with third order numerical quadrature rules, cf. Appendix B.

Euler flux terms

The first integral argument in the weak form (4.3) stems from the Euler fluxes and the time derivative of the conserved variables (here written as a time flux $F_0^\mathsf{e}(U) = U$),

$$\mathfrak{A}^n(V, W) := -\int_{\mathcal{E}_h^n} W_{,r} \cdot F_r^\mathsf{e} \, \mathrm{d}\mathcal{E} \tag{4.9}$$

$$= -\sum_{e=1}^{N_\mathsf{el}^n} \int_{\hat{\mathcal{K}}_e^n} \left(\frac{2}{\Delta t_n} \frac{\partial W}{\partial \hat{x}_0} F_0^\mathsf{e}(V(\hat{x})) + \frac{\partial \hat{x}_{\bar{s}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{s}}} F_{\bar{r}}^\mathsf{e}(V(\hat{x})) \right) \left| \mathrm{Jac}_{\hat{x}} G_e^n \right| \, \mathrm{d}\hat{\mathcal{K}} \tag{4.10}$$

$$= \frac{\Delta t_n}{2} \sum_{e=1}^{N_\mathsf{el}^n} \int_{\hat{\mathrm{T}}} A_e^n(V, W, \zeta) \, \mathrm{d}\zeta \tag{4.11}$$

$$\overset{\mathsf{TR}}{\approx} \frac{\Delta t_n}{2} \sum_{e=1}^{N_\mathsf{el}^n} \sum_{j=1}^{2} A_e^{n,j}(V, W) \,, \tag{4.12}$$

with the space integral A_e^n on the element $\mathcal{K}_e^n$ defined as

$$A_e^n(V,W,\zeta) := -\int\limits_{\hat{\bar{\mathcal{K}}}_e^n} \left(\frac{2}{\Delta t_n} \frac{\partial W}{\partial \hat{x}_0}(\zeta)\, F_0^{\mathsf{e}}(V(\zeta)) + \frac{\partial \hat{x}_{\bar{s}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{s}}}(\zeta)\, F_{\bar{r}}^{\mathsf{e}}(V(\zeta)) \right) \left|\mathrm{Jac}_{\hat{\bar{x}}} \bar{G}_e^n(\zeta)\right| \,\mathrm{d}\hat{\bar{\mathcal{K}}}\,, \tag{4.13}$$

and the shorthand notation $A_e^{n,j}(V,W) := A_e^n(V,W,\hat{x}_0^j)$ for the integrals at the time interval end values $\hat{x}_0^1 = -1$ and $\hat{x}_0^2 = 1$. The mapping $\bar{G}_e^n(\zeta)$ maps from the space reference element $\hat{\bar{\mathcal{K}}}_e^n$ to the physical space element at time t, $\bar{\mathcal{K}}_e^n$, where $t = G_0^n(\zeta)$. For the constant in time expansion, the summation argument in (4.12) yields the same result for both values of j. The Jacobian with respect to the local expansion coefficients is

$$\frac{\partial A_e^{n,j}}{\partial V_{i',e'}^{n',j'}} = -\delta_{ee'}\delta_{nn'} \int\limits_{\hat{\bar{\mathcal{K}}}_e^n} \left(\frac{2}{\Delta t_n} \frac{\partial W}{\partial \hat{x}_0}(\hat{x}_0^j)\, A_0^V(V_e^{n,j}) + \frac{\partial \hat{x}_{\bar{s}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{s}}}(\hat{x}_0^j)\, A_{\bar{r}}^V(V_e^{n,j}) \right) \hat{\lambda}^{j'}(\hat{x}_0^j)\, \hat{\psi}_{i'} \left|\mathrm{Jac}_{\hat{\bar{x}}} \bar{G}_e^n(\hat{x}_0^j)\right| \,\mathrm{d}\hat{\bar{\mathcal{K}}}\,. \tag{4.14}$$

Here, the evaluation of the trial function at the time slab end j is denoted $V_e^{n,j} := V_e^n(\hat{x}_0^j)$. Obviously $\partial A_e^{n,j}/\partial V_{i',e'}^{n',j'}$ only gives a contribution when $n = n'$ and $e = e'$, i.e., when the degree of freedom with respect to which the expression is linearized is associated with the element $\mathcal{K}_e^n$.

Viscous term

The weak formulation (4.3) exploits that the diffusive fluxes are homogeneous functions of order one in the conserved variables, so that—transformed to entropy variables—the related term is

$$\mathfrak{B}^n(V,W) := \int\limits_{\mathcal{E}_h^n} W_{,\bar{r}} \cdot (K_{\bar{r}\bar{s}}^V V_{,\bar{s}})\,\mathrm{d}\mathcal{E} \tag{4.15}$$

$$= \sum_{e=1}^{N_{\mathsf{el}}^n} \int\limits_{\hat{\mathcal{K}}_e^n} \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{p}}} \cdot \left(K_{\bar{r}\bar{s}}^V \frac{\partial \hat{x}_{\bar{q}}}{\partial x_{\bar{s}}} \frac{\partial V}{\partial \hat{x}_{\bar{q}}} \right) \left|\mathrm{Jac}_{\hat{x}} G_e^n\right| \,\mathrm{d}\hat{\mathcal{K}} \tag{4.16}$$

$$= \frac{\Delta t_n}{2} \sum_{e=1}^{N_{\mathsf{el}}^n} \int\limits_{\hat{\mathrm{T}}} B_e^n(V,W,\zeta)\,\mathrm{d}\zeta \tag{4.17}$$

$$\overset{\mathsf{TR}}{\approx} \frac{\Delta t_n}{2} \sum_{e=1}^{N_{\mathsf{el}}^n} \sum_{j=1}^{2} B_e^{n,j}(V,W)\,, \tag{4.18}$$

with the space integral B_e^n defined as (omitting the indication for evaluation at reference time ζ in favor of shorter notation)

$$B_e^n(V, W, \zeta) := \int\limits_{\hat{\hat{\mathcal{K}}}_e^n} \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{p}}} \cdot \left(K_{\bar{r}\bar{s}}^V \frac{\partial \hat{x}_{\bar{q}}}{\partial x_{\bar{s}}} \frac{\partial V}{\partial \hat{x}_{\bar{q}}} \right) \left| \mathrm{Jac}_{\hat{\hat{x}}} \bar{G}_e^n(\zeta) \right| \, \mathrm{d}\hat{\hat{\mathcal{K}}}, \tag{4.19}$$

and the notation for its evaluation at time level j, $B_e^{n,j}(V, W) := B_e^n(V, W, \hat{x}_0^j)$. For the Jacobian of this term, the dependence of the matrices $K_{\bar{i}\bar{j}}^V$ on the local state V is neglected, so that the linearization with respect to the expansion coefficients can be expressed as

$$\frac{\partial B_e^{n,j}}{\partial V_{i',e'}^{n',j'}} = \delta_{ee'} \delta_{nn'} \int\limits_{\hat{\hat{\mathcal{K}}}_e^n} \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{p}}} \cdot \left(K_{\bar{r}\bar{s}}^V \frac{\partial \hat{x}_{\bar{q}}}{\partial x_{\bar{s}}} \hat{\lambda}^{j'}(\hat{x}_0^j) \frac{\partial \hat{\psi}_{i'}}{\partial \hat{x}_{\bar{q}}} \right) \left| \mathrm{Jac}_{\hat{\hat{x}}} \bar{G}_e^n(\hat{x}_0^j) \right| \, \mathrm{d}\hat{\hat{\mathcal{K}}}. \tag{4.20}$$

Least-squares term

Using the quasi-linear form of the Navier–Stokes operator $\mathcal{L}_V$,

$$\mathcal{L}_V := A_r^V \frac{\partial}{\partial x_r} - \frac{\partial}{\partial x_{\bar{r}}} \left(K_{\bar{r}\bar{s}}^V \frac{\partial}{\partial x_{\bar{s}}} \right), \tag{4.21}$$

which is symmetric for entropy variables V, and the stabilization operator τ_V (see Section 4.4), the least-squares term is defined as

$$\mathfrak{C}^n(V, W) \equiv B_{ls}(V, W) := \int\limits_{\mathcal{E}_h^n} \mathcal{L}_V W \cdot \tau_V \, \mathcal{L}_V V \, \mathrm{d}\mathcal{E} \tag{4.22}$$

$$= \sum_{e=1}^{N_{\mathrm{el}}^n} \int\limits_{\hat{\mathcal{K}}_e^n} \mathcal{L}_V W \cdot \tau_V \, \mathcal{L}_V V \left| \mathrm{Jac}_{\hat{x}} G_e^n \right| \, \mathrm{d}\hat{\mathcal{K}} \tag{4.23}$$

$$= \frac{\Delta t_n}{2} \sum_{e=1}^{N_{\mathrm{el}}^n} \int\limits_{\hat{\mathrm{T}}} C_e^n(V, W, \zeta) \, \mathrm{d}\zeta \tag{4.24}$$

$$\overset{\mathrm{TR}}{\approx} \frac{\Delta t_n}{2} \sum_{e=1}^{N_{\mathrm{el}}^n} \sum_{j=1}^{2} C_e^{n,j}(V, W). \tag{4.25}$$

Here, as before, the integration is split into the time integral, which has been approximated with the trapezoidal rule above, and the space integral of the integrand, C_e^n,

$$
\begin{aligned}
C_e^n(V,W,\zeta) := \int\limits_{\hat{\hat{\mathcal{K}}}_e^n} &\left(\frac{2}{\Delta t_n} A_0^V \frac{\partial W}{\partial \hat{x}_0} + A_{\bar{r}}^V \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{p}}} + \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial}{\partial \hat{x}_{\bar{p}}} \left(K_{\bar{r}\bar{s}}^V \frac{\partial \hat{x}_{\bar{q}}}{\partial x_{\bar{s}}} \frac{\partial W}{\partial \hat{x}_{\bar{q}}} \right) \right) \\
&\cdot \tau_V \left(\frac{2}{\Delta t_n} A_0^V \frac{\partial V}{\partial \hat{x}_0} + A_{\bar{r}'}^V \frac{\partial \hat{x}_{\bar{p}'}}{\partial x_{\bar{r}'}} \frac{\partial V}{\partial \hat{x}_{\bar{p}'}} + \frac{\partial \hat{x}_{\bar{p}'}}{\partial x_{\bar{r}'}} \frac{\partial}{\partial \hat{x}_{\bar{p}'}} \left(K_{\bar{r}'\bar{s}'}^V \frac{\partial \hat{x}_{\bar{q}'}}{\partial x_{\bar{s}'}} \frac{\partial V}{\partial \hat{x}_{\bar{q}'}} \right) \right) \\
&\left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}_e^n(\zeta)\right| \,\mathrm{d}\hat{\hat{\mathcal{K}}}\,.
\end{aligned}
\tag{4.26}
$$

The matrices A_i^V and $K_{\bar{i}\bar{j}}^V$ depend on the state V and thus implicitly also on the space coordinates. For practical reasons, however, for this term the $K_{\bar{i}\bar{j}}^V$-matrices are assumed constant per element. The Jacobian is

$$
\begin{aligned}
\frac{\partial C_e^{n,j}}{\partial V_{i',e'}^{n',j'}} = \delta_{ee'}\delta_{nn'} \int\limits_{\hat{\mathcal{K}}_e^n} &\left(\frac{2}{\Delta t_n} A_0^V \frac{\partial W}{\partial \hat{x}_0} + A_{\bar{r}}^V \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial W}{\partial \hat{x}_{\bar{p}}} + \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{r}}} \frac{\partial}{\partial \hat{x}_{\bar{p}}} \left(K_{\bar{r}\bar{s}}^V \frac{\partial \hat{x}_{\bar{q}}}{\partial x_{\bar{s}}} \frac{\partial W}{\partial \hat{x}_{\bar{q}}} \right) \right) \\
&\cdot \tau_V \Bigg(\frac{2}{\Delta t_n} A_0^V \underline{\frac{\partial \hat{\lambda}^{j'}}{\partial \hat{x}_0}} (\hat{x}_0^j)\, \hat{\psi}_{i'} + A_{\bar{r}'}^V \hat{\lambda}^{j'}(\hat{x}_0^j) \frac{\partial \hat{x}_{\bar{p}'}}{\partial x_{\bar{r}'}} \frac{\partial \hat{\psi}_{i'}}{\partial \hat{x}_{\bar{p}'}} \\
&\qquad + \hat{\lambda}^{j'} \frac{\partial \hat{x}_{\bar{p}'}}{\partial x_{\bar{r}'}} \underbrace{\frac{\partial}{\partial \hat{x}_{\bar{p}'}} \left(K_{\bar{r}'\bar{s}'}^V \frac{\partial \hat{x}_{\bar{q}'}}{\partial x_{\bar{s}'}} \frac{\partial \hat{\psi}_{i'}}{\partial \hat{x}_{\bar{q}'}} \right)}_{*} \Bigg) \left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}_e^n(\hat{x}_0^j)\right| \,\mathrm{d}\hat{\Omega}\,.
\end{aligned}
\tag{4.27}
$$

Shakib et al. (1991) advise to discard the underlined term due to a destabilizing effect,[1] see also (Polner, 2005). Further, the term marked with $*$ is approximated as

$$
\frac{\partial}{\partial \hat{x}_{\bar{p}'}} \left(K_{\bar{r}'\bar{s}'}^V \frac{\partial \hat{x}_{\bar{q}'}}{\partial x_{\bar{s}'}} \frac{\partial \hat{\psi}_{i'}}{\partial \hat{x}_{\bar{q}'}} \right) \approx K_{\bar{r}'\bar{s}'}^V \frac{\partial \hat{x}_{\bar{q}'}}{\partial x_{\bar{s}'}} \frac{\partial^2 \hat{\psi}_{i'}}{\partial \hat{x}_{\bar{p}'} \partial \hat{x}_{\bar{q}'}}\,.
\tag{4.28}
$$

Boundary conditions

When applying the Gauß theorem, integrals over the discrete space-time boundary Q_h^n of the n^{th} slab enter the weak formulation:

$$
\mathfrak{D}^n(V,W) \equiv B_{\mathsf{bc}}(V,W) \tag{4.29}
$$

$$
= \sum_{e=1}^{N_{\mathsf{el}}^n} \sum_{\substack{k \\ F_{e,k}^n \cap Q_h^n \neq \emptyset}} \int\limits_{F_{e,k}^n} \left((W \cdot F_r^{\mathsf{e}}(V))\, n_r - W \cdot (K_{r\bar{s}}^V V_{,\bar{s}})\, n_r \right) \mathrm{d}F \tag{4.30}
$$

[1]The term also yields a contribution even if $j \neq j'$, so that when computing the Jacobian, the sum over j would have to be carried out.

$$
\begin{aligned}
&= \sum_{e=1}^{N_{\mathsf{el}}^n} \sum_{\substack{k \\ F_{e,k}^n \cap Q_h^n \neq \emptyset}} \int_{\hat{\mathrm{T}}} \int_{\hat{\bar{F}}_{e,k}^n} \left\{ (W \cdot F_r^{\mathsf{e}}(V(\hat{x})))\, n_r - W \cdot K_{r\bar{s}}^V \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{s}}} \frac{\partial V}{\partial \hat{x}_{\bar{p}}}\, n_r \right\} \\
&\qquad\qquad \left| \mathrm{Jac}_{\hat{\bar{x}}} G_{\hat{\bar{F}}_{e,k}^n}^{\bar{F}_{e,k}^n} \right| \underbrace{\left| \mathrm{Jac}_{\hat{x}_0} G_{\hat{\mathrm{T}}}^{\mathrm{T}^n} \right|}_{\Delta t_n / 2} \,\mathrm{d}\hat{\bar{F}}\, \mathrm{d}\zeta
\end{aligned}
\tag{4.31}
$$

$$
\overset{\mathsf{TR}}{\approx} \frac{\Delta t_n}{2} \sum_{e=1}^{N_{\mathsf{el}}^n} \sum_{j=1}^{2} D_e^{n,j}(V, W)\,, \tag{4.32}
$$

where the coordinates of the space reference face were denoted $\hat{\bar{x}}$.

Remark 4.1 *For moving meshes, the time component of the normal vector on the space-time boundary, n_0, is nonzero, but the diffusive flux in the time direction vanishes, which is reflected by $K_{0\bar{\imath}}^V = 0$.* □

The summation argument is given by the integral $D_e^{n,j}(W) := D_e^n(V, W, \hat{x}_0^j)$ with

$$
\begin{aligned}
D_e^n(V, W, \zeta) := \sum_{\substack{k \\ F_{e,k}^n \cap Q_h^n \neq \emptyset}} \int_{\hat{\bar{F}}_{e,k}^n} \Big\{ &(W \cdot F_r^{\mathsf{e}}(V(\hat{x})))\, n_r \\
&- W \cdot K_{r\bar{s}}^V \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{s}}} \frac{\partial V}{\partial \hat{x}_{\bar{p}}} n_r \Big\} \left| \mathrm{Jac}_{\hat{\bar{x}}} G_{\hat{\bar{F}}_{e,k}^n}^{\bar{F}_{e,k}^n} \right| \mathrm{d}\hat{\bar{F}}\,.
\end{aligned}
\tag{4.33}
$$

The linearization of this term is

$$
\begin{aligned}
\frac{\partial D_e^{n,j}}{\partial V_{i',e'}^{n',j'}} = \delta_{ee'} \delta_{nn'} \sum_{\substack{k \\ F_{e,k}^n \cap Q_h^n \neq \emptyset}} \int_{\hat{\bar{F}}_{e,k}^n} \Big\{ &(W \cdot A_r^V(V))\, n_r\, \hat{\lambda}^{j'}(\hat{x}_0^j)\, \hat{\psi}_{i'}(\hat{\bar{x}}) \\
&- W \cdot K_{r\bar{s}}^V \frac{\partial \hat{x}_{\bar{p}}}{\partial x_{\bar{s}}} \frac{\partial \hat{\psi}_{i'}}{\partial \hat{x}_{\bar{p}}} \hat{\lambda}^{j'}(\hat{x}_0^j)\, n_r \Big\} \left| \mathrm{Jac}_{\hat{\bar{x}}} G_{\hat{\bar{F}}_{e,k}^n}^{\bar{F}_{e,k}^n} \right| \mathrm{d}\hat{\bar{F}}\,.
\end{aligned}
\tag{4.34}
$$

The basis function $\hat{\psi}_i$ has to be evaluated on the appropriate (reference element) boundary over which the integral is taken.

Time face terms

Across the time slab boundaries the time basis functions are discontinuous, but continuity of the solution is enforced in a weak sense by the integrals on the time faces:

$$
\mathfrak{E}^n(V, W) := \int_{\Omega(t_n)} W(t_n^-, \bar{x}) \cdot F_0^{\mathsf{e}}(V(t_n^-, \bar{x}))\, \mathrm{d}\Omega - \int_{\Omega(t_{n-1})} W(t_{n-1}^+, \bar{x}) \cdot F_0^{\mathsf{e}}(V(t_{n-1}^-, \bar{x}))\, \mathrm{d}\Omega \tag{4.35}
$$

$$\begin{aligned} &= \sum_{e=1}^{N^n_{\mathsf{el}}} \int\limits_{\hat{\hat{\mathcal{K}}}^n_e} W(\hat{x}_0 = 1, \hat{\hat{x}}) \cdot F^{\mathsf{e}}_0(V^n_e(\hat{x}_0 = 1, \hat{\hat{x}})) \left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}^{n,-}_e\right| \,\mathrm{d}\hat{\hat{\mathcal{K}}} \\ &\qquad - \sum_{e=1}^{N^n_{\mathsf{el}}} \int\limits_{\hat{\hat{\mathcal{K}}}^n_e} W(\hat{x}_0 = -1, \hat{\hat{x}}) \cdot F^{\mathsf{e}}_0(V^{n-1}_{e'}(\hat{x}_0 = 1, \hat{\hat{x}})) \left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}^{n-1,+}_e\right| \,\mathrm{d}\hat{\hat{\mathcal{K}}} \end{aligned} \tag{4.36}$$

$$= \sum_{e=1}^{N^n_{\mathsf{el}}} \sum_{j=1}^{2} E^{n,j}_e(V, W)\,. \tag{4.37}$$

Notably the data at the beginning of the time step is determined by the state at the end of the previous time slab, hence $V^{n-1}_{e'}(\hat{x}_0 = 1)$. This data is not necessarily based on the element with the same index e as the one in the current slab, since the mesh may be changed over the time slab boundary. The above summation argument is defined as

$$E^{n,j}_e(V, W) := \begin{cases} -\int_{\hat{\hat{\mathcal{K}}}^n_e} W(\hat{x}^1_0) F^{\mathsf{e}}_0(V^{n-1}_e(\hat{x}^2_0, \hat{\hat{x}})) \left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}^{n-1,+}_e\right| \,\mathrm{d}\hat{\hat{\mathcal{K}}} & \text{if } j = 1\,, \\ \int_{\hat{\hat{\mathcal{K}}}^n_e} W(\hat{x}^2_0) F^{\mathsf{e}}_0(V^n_e(\hat{x}^2_0, \hat{\hat{x}})) \left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}^{n,-}_e\right| \,\mathrm{d}\hat{\hat{\mathcal{K}}} & \text{if } j = 2\,, \end{cases} \tag{4.38}$$

and its Jacobian is

$$\frac{\partial E^{n,j}_k}{\partial V^{n',j'}_{i',e'}} = \begin{cases} 0 & \text{if } j = 1\,, \\ \delta_{ee'} \delta_{nn'} \int_{\hat{\hat{\mathcal{K}}}^n_e} W(\hat{x}^2_0)\, A^V_0(V^n_e(\hat{x}^2_0, \hat{\hat{x}}))\, \hat{\psi}_{i'}\, \hat{\lambda}^{j'}(\hat{x}^j_0) \left|\mathrm{Jac}_{\hat{\hat{x}}} \bar{G}^{n,-}_e\right| \,\mathrm{d}\hat{\hat{\mathcal{K}}} & \text{if } j = 2\,. \end{cases} \tag{4.39}$$

The first case evaluates to zero because this contribution stems from the derivative of the state in slab $\mathcal{E}^{n-1}_h$ with respect to the degrees of freedom of the current (n^{th}) space-time slab.

Nonlinear system of equations

By choosing the (global) test functions as $W^n_{1,g} = \lambda^{n,1}\psi^{n,g}$, and $W^n_{2,g} = \lambda^{n,2}\psi^{n,g}$, a global system of equations results as

$$R^{n,1}_g(V) := \mathfrak{A}^n(V, W^{n,1}_g) + \ldots + \mathfrak{E}^n(V, W^{n,1}_g) = 0\,, \tag{4.40a}$$

$$R^{n,2}_g(V) := \mathfrak{A}^n(V, W^{n,2}_g) + \ldots + \mathfrak{E}^n(V, W^{n,2}_g) = 0\,. \tag{4.40b}$$

The system is nonlinear, but using the linearization carried out before, a Newton method can be applied to obtain its solution. In this case, the linear system that is solved in every step of the Newton iteration for the updates $\Delta V^{n,i}_g$ of the expansion coefficients with respect to the time- and space basis function combination $\lambda^{n,i}\psi^{n,g}$, $i \in \{1, 2\}$, has a block structure of the form

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} \Delta V^{n,1} \\ \Delta V^{n,2} \end{pmatrix} = - \begin{pmatrix} R^{n,1} \\ R^{n,2} \end{pmatrix} , \tag{4.41}$$

with the blocks

$$\left(M_{\alpha\beta}\right)_{ij} = \frac{\partial R_i^{n,\alpha}}{\partial V_j^{n,\beta}} , \quad \alpha,\beta \in \{1,2\}, \quad i,j \in \{1,\dots,\#\text{d.o.f.}\} , \tag{4.42}$$

where i and j enumerate the degrees of freedom of the discrete representation of V. To reduce the computational effort, Polner (2005) discusses a simplification of the Newton algorithm making use of the system matrix constructed symbolically as

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} M_{11} + M_{12} & -M_{11} + M_{12} \\ M_{21} + M_{22} & -M_{21} + M_{22} \end{pmatrix} , \tag{4.43}$$

which is approximated well by

$$\begin{pmatrix} 2M_{11} & 0 \\ 0 & 2M_{22} \end{pmatrix} , \tag{4.44}$$

and enables to solve in the Newton procedure with uncoupled blocks for updates for the means $\bar{V}^n := \frac{1}{2}(V^{n,1} + V^{n,2})$ and differences $\overset{\vartriangle}{V}{}^n := \frac{1}{2}(-V^{n,1} + V^{n,2})$, as follows from

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \Delta V^{n,1} \\ \Delta V^{n,2} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \Delta V^{n,1} \\ \Delta V^{n,2} \end{pmatrix} = \begin{pmatrix} \Delta \bar{V}^n \\ \Delta \overset{\vartriangle}{V}{}^n \end{pmatrix} . \tag{4.45}$$

The change of variables is equivalent to choosing the test functions as $W_{1,g}^n = \psi^{n,g}$, and $W_{2,g}^n = (\lambda^{n,1} - \lambda^{n,2})\psi^{n,g} = (t_{n-1} + t_n - 2t)/(t_n - t_{n-1})\,\psi^{n,g}$.

For steady-state computations, the described procedure can be simplified by using only a constant-in-time approximation (Shakib et al., 1991). In this case, the system is reduced to the size of one of the above blocks, since only $\bar{V}^n$ needs to be solved for.

4.3 Boundary conditions

The enforcement of some types of boundary conditions is intricate in continuous Galerkin methods. The boundary integral in the weak form (4.3) can accommodate Neumann boundary conditions directly, but these neither necessarily lead to a mathematically well-posed formulation nor are they physically relevant for all variables. When no Neumann boundary conditions are used, the boundary term $\mathfrak{D}^n(V, W)$ is computed from the data of the current approximation and the relevant conditions have to be enforced in a different way. Shakib et al. (1991) advocate prescribing Dirichlet conditions via constraint equations. Alternatively, to specify a Dirichlet boundary condition for a certain global degree of freedom j_{dof}, the computed entry in the vector of expansion coefficients can be overwritten with the given value, the row j_{dof} in the diagonal block of the system (4.41) is filled with zeros, and the entry at the position $(j_{\mathsf{dof}}, j_{\mathsf{dof}})$ is set to one. Additionally, the row j_{dof} of the horizontally neighboring non-diagonal matrix block and the right hand side vector are overwritten by zeros. The results presented in this chapter use the latter method for prescribing Dirichlet boundary values. In comparison with linearized constraints, which were used before, there was no observable difference between the two methods.

Another issue regarding boundary conditions concerns the use of entropy variables in the global system, while many boundary conditions (whether of Dirichlet or Neumann type) concern primitive (or other) variables. For instance, on the inflow boundary one often specifies values for the velocity components and temperature. Also on no-slip boundaries velocities are imposed. On outflow boundaries, pressure (or rather: the normal stress, which for high Reynolds numbers is dominated by the pressure) and vanishing tangential stress are prescribed. Since there is no correspondence by components between entropy and primitive variables, it is not a priori clear how to impose a primitive variable boundary condition on an entropy variable solution. For the previously mentioned cases, the following procedure is followed: If, in a boundary node, one or more components require imposing a primitive variable boundary condition, then the current entropy variable approximation is transformed to primitive variables and the given conditions are imposed. Next, the changed vector of primitive variables is transformed back to entropy variables and overwrites the previously computed approximation. Of course, in the assembly of the global system (4.41), still only those matrix lines are changed that belong to components that have a specified Dirichlet condition.

4.4 Least-squares stabilization operator

In the above derivations, the least-squares term includes the so far unspecified stabilization matrix τ_V. The basic requirements on τ_V arising from (4.3) are symmetry, positive definiteness, appropriate scaling with the element size, and dimensional consistency (Polner, Pesch, and van der Vegt, 2007). These properties are central for the stabilizing effect, which the least-squares term is intended to have. At the same time, it should not compromise the accuracy of the discretization. The cited requirements are not sufficient to deduce a unique functional form of τ_V. Therefore various stabilization matrices have been proposed. In the context of the entropy/general variable approach, Hauke and Hughes (1998) experimentally designed a stabilization matrix. Hauke (2001) surveyed the available choices for computing incompressible flows with primitive variables and compressible flows with conserved variables; he points out the lack of stabilization matrices for compressible flows when other than conserved variables are used. Hauke proposes a diagonal stabilization matrix, which stabilizes each component of the Navier–Stokes equation individually, as in the one-dimensional theory. Because the diagonal operator lacks robustness and is undefined in the incompressible limit, he examined also a non-diagonal matrix, which was found to be more robust than the diagonal version and computationally cheaper than the matrix used in previous work, e.g., (Shakib et al., 1991; Hauke and Hughes, 1998). Unfortunately, since the definition of the non-diagonal matrix uses primitive variables and the Jacobian $Y_{,U}$, the problem regarding the incompressible limit remained unsolved.

In (Polner et al., 2007), great care is taken to provide the stabilization term with an additional property of the entropy-variable formulation, namely the well-defined incompressible limit. Polner et al. design a stabilization matrix for compressible and incompressible flow starting from the dimensional analysis of the governing equations. This fixes the dimensions of the entries of τ_V. Next, the analysis of the low Mach-number limit of the Galerkin operator, and the requirement that the least-squares operator has the same asymptotic behavior, relates the limiting behavior of the entries of the matrix to the compressibility parameters. Based on these findings, a functional form of τ_V, is proposed. The necessary and sufficient condition to ensure nonlinear stability of the Galerkin least-squares method is the symmetric positive definiteness of the stabilization matrix for entropy variables, which yields further information about the matrix entries.

These requirements are initially imposed in the primitive variable formulation and yield the class of dimensionally consistent stabilization matrices τ_Y, with a well-defined incompressible limit as $\alpha_p \to 0$ in the form

$$\tau_Y = \begin{pmatrix} \tau_{\mathrm{c}} & \rho(\omega+1)\left(\tau_{\mathrm{m}} + (h-k)\alpha_p\tau_{\mathrm{e}}\right)v_{\bar{\jmath}} & 2\rho\alpha_p\tau_{\mathrm{e}}k \\ \omega\left(\tau_{\mathrm{m}} + (h-k)\alpha_p\tau_{\mathrm{e}}\right)v_{\bar{\imath}} & \delta_{\bar{\imath}\bar{\jmath}}\tau_{\mathrm{m}} & \alpha_p\tau_{\mathrm{e}}v_{\bar{\imath}} \\ -(h-k)\tau_{\mathrm{e}} & (\alpha_pT-1)\tau_{\mathrm{e}}v_{\bar{\jmath}} & \tau_{\mathrm{e}} \end{pmatrix}, \tag{4.46}$$

where $v_{\bar{\imath}}$ are the velocity components, h the specific enthalpy, and $k = v_{\bar{r}}^2/2$ the specific kinetic energy. The parameter $\omega \in \mathbb{R}$ is bounded by the requirement of positive definiteness of τ_V. For an analysis of the range of admissible values for ω see (Polner, 2005). The stabilization parameters are defined as

$$\tau_{\mathrm{c}} = \frac{h_{\mathrm{el}}\,|\boldsymbol{v}|}{2}, \quad \tau_{\mathrm{m}} = \frac{h_{\mathrm{el}}}{2\rho|\boldsymbol{v}|}\ell(\mathrm{Re}_{\mathrm{el}}), \quad \tau_{\mathrm{e}} = \frac{\tau_{\mathrm{m}}}{c_{\mathrm{v}}}, \tag{4.47}$$

with the element diameter h_{el}. The *element Reynolds number* $\mathrm{Re}_{\mathrm{el}}$ is defined as

$$\mathrm{Re}_{\mathrm{el}} = \frac{m_k\,\rho\,|\boldsymbol{v}|\,h_{\mathrm{el}}}{\eta}, \tag{4.48}$$

with $m_k = \min\{1/3, 2C_k\}$, where C_k is a positive, mesh- and state-independent constant related to the maximum polynomial degree of the basis functions used on the element. Here, $m_k = 1/3$ is prescribed, cf. (Franca et al., 1992). Finally, $\ell\colon \mathbb{R}_0^+ \to [0;1]$ is a limiter function defined as

$$\ell(v) := \begin{cases} v, & \text{if} \quad 0 \le v < 1\,, \\ 1, & \text{otherwise}\,. \end{cases} \tag{4.49}$$

To obtain a stabilization matrix for entropy variables, the abovementioned operator τ_Y is transformed by multiplying the stabilization matrix for pressure primitive variables with the Jacobian of the variable transformation $V(Y)$, i.e., $\tau_V = V_{,Y}\tau_Y$. The resulting operator has the form

$$\tau_V = \begin{pmatrix} \frac{1}{T}\left(\alpha\tau_{\mathsf{c}} - \omega v_{\bar{r}}^2\tau_{\mathsf{m}} + \frac{1}{T}(h-k)^2\,\tau_{\mathsf{e}} - \omega v_{\bar{r}}^2(h-k)\tau_{\mathsf{e}}\alpha_p\right) & & \text{sym.} \\ \frac{1}{T}\left(\omega\tau_{\mathsf{m}} + (h-k)\tau_{\mathsf{e}}\left(\frac{1}{T} + \omega\alpha_p\right)\right)v_{\bar{\imath}} & \frac{1}{T}\left(\tau_{\mathsf{m}}\delta_{\bar{\imath}\bar{\jmath}} + v_{\bar{\imath}}\,v_{\bar{\jmath}}\tau_{\mathsf{e}}\left(\frac{1}{T} - \alpha_p\right)\right) & \\ -\frac{1}{T^2}(h-k)\tau_{\mathsf{e}} & -\frac{1}{T}v_{\bar{\jmath}}\tau_{\mathsf{e}}\left(\frac{1}{T} - \alpha_p\right) & \frac{1}{T^2}\tau_{\mathsf{e}} \end{pmatrix}. \tag{4.50}$$

4.5 Numerical experiments

In this section, results of computations for several flow problems using the stabilization operator τ_V are presented. The key feature of this stabilization operator is that it applies to both compressible and incompressible fluids. Both cases are presented alongside in (Polner et al., 2007), from which the results for the compressible case are included here.

The discretization uses the Galerkin least-squares formulation given in Section 4.2 with entropy variables, linear basis functions in space for all variables and the constant-in-time approximation as presented by Shakib et al. (1991).

4.5.1 Poiseuille flow

In this test case, flow along a channel with no-slip walls is considered. Poiseuille flow is an exact solution of the incompressible Navier–Stokes equations and therefore a classical problem to verify numerical methods. The resulting flow is aligned with the channel, the velocity in that direction having a parabolic profile, and the pressure is increasing linearly in the counter-stream direction. Dissipation due to shear creates a fourth order parabolic temperature profile across the channel. Note that the described exact solution holds only for an incompressible fluid. For a compressible medium its thermodynamic properties couple the thermodynamic state back to the flow.

Although the exact solution assumes the channel to be infinitely long, the simulations are based on a finite length domain $\Omega = [0; 2] \times [0; 1]$ with non-periodic boundaries. At the inflow, the parabolic velocity profile of the exact solution for incompressible flow and a constant temperature are prescribed as Dirichlet conditions. At the outflow, only the pressure is specified. The walls are no-slip with constant temperature. All computations use time-dependent formulations and are run until a steady state is reached. Two values of the Reynolds number are considered: $\mathrm{Re}_\eta = 1$ and $\mathrm{Re}_\eta = 100$. The Eckert number, which occurs in the non-dimensional energy equation, is set to $\mathrm{Ec} = 0.016$, which corresponds to

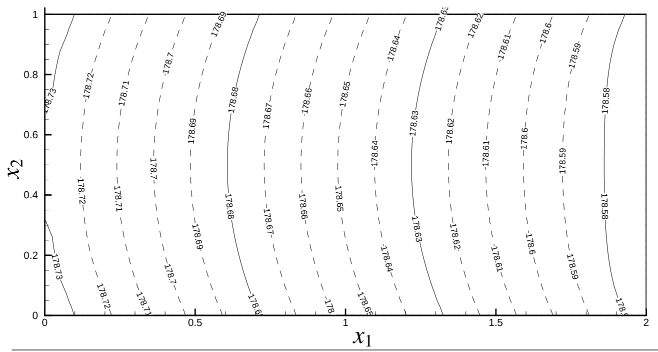


Figure 4.1: Pressure field for the compressible Poiseuille flow computed on an 80×40 mesh with $\mathrm{Re}_\eta = 100$. The flow direction is from left to right.

a Mach number of $M = 0.2$ for the compressible flow of an ideal, diatomic (i.e., $\gamma = 7/5$) gas with constant specific heats. The resulting equations of state are given by (2.30a) and (2.30b) and the Prandtl number is set to $\mathrm{Pr} = 0.715$, a typical value for air. A converged solution on an 80×40 grid serves as reference for the error computation.

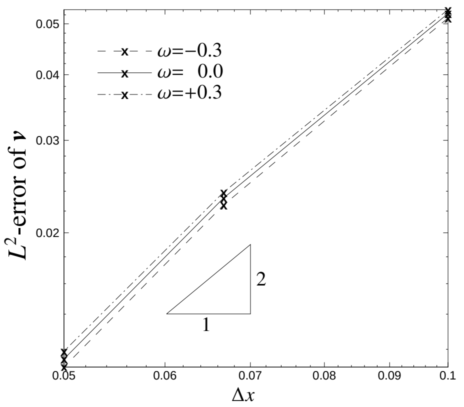
The aim of this case is to test whether the stabilization matrix is effective for compressible flows. It is verified that the least-squares operator does not affect the second order spatial accuracy of the method, when linear polynomial basis functions are used. The accuracy is measured by the $L^2(\Omega)$ norm of the velocity field.

Results

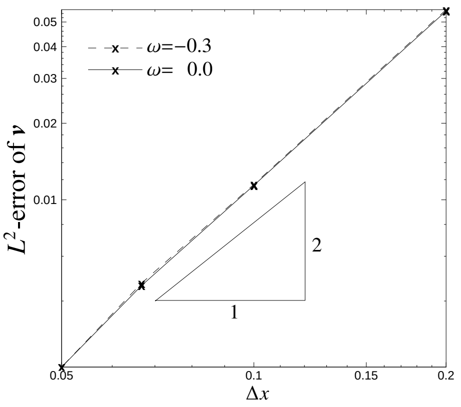
The velocity field is virtually the same as for the incompressible case, cf. (Polner et al., 2007). As expected, the pressure in the compressible fluid adapts differently to the flow state than in the incompressible case, see Figure 4.1. Figure 4.2 documents the second order accuracy of the method; furthermore, it confirms that different values of the parameter ω in the stabilization matrix do not degrade the accuracy of the method. On the other hand, the stabilization parameter ω may influence the convergence of the iterative solver, as was noticed when monitoring the residual norm during the computation on a fixed mesh, cf. (Polner, 2005).

4.5.2 Driven cavity flow

The driven cavity flow is a classical problem to test algorithms for incompressible flows. The fluid is enclosed in a unit square domain with no-slip walls, one of which is moving with a constant velocity $u_1 = 1$ in the tangential direction. At the contact points of this moving lid with the neighboring boundaries the velocity is discontinuous. The challenge of the test case is to control the singularities in these points while accurately representing the smooth regions of the flow, away from the lid. The use of the stabilization operator is essential for both compressible and incompressible flows to ensure stability without compromising accuracy. A clustered mesh is used to resolve the discontinuities in the two corners at the lid while being able to compute the driven cavity flow on meshes with



(a) $\mathrm{Re}_\eta = 1$.



(b) $\mathrm{Re}_\eta = 100$.

Figure 4.2: Compressible Poiseuille flow, error of the velocity field (measured in the $L^2(\Omega)$-norm) as a function of the mesh size Δx for different values of the stabilization parameter ω. Two Reynolds numbers are used: (a) $\mathrm{Re}_\eta = 1$, and (b) $\mathrm{Re}_\eta = 100$; for the higher Reynolds number only two ω values are compared, as for $\omega = +0.3$ results could only be obtained with at least a 40×20 mesh.

relatively few elements. The dimensionless parameters Ec, Pr, and γ are chosen as in Section 4.5.1. The computations are initialized with zero velocities and constant pressure and temperature. The boundary conditions are no-slip for the velocity and Dirichlet-type for the temperature.

Although the driven cavity is not typically used as a test case for compressible flow algorithms, to correspond with the incompressible test results for the stabilization operator presented in (Polner et al., 2007), the computation of the driven cavity flow has been repeated for compressible flow at $\mathrm{Re}_\eta = 400$.

Results

Figure 4.3 on the following page shows the computational mesh, streamlines, and the temperature field of the developed compressible flow. In this computation the new stabilization matrix for entropy variables was employed, with the stabilization parameter set to $\omega = 0$. Qualitative agreement with the numerical results of Ghia et al. (1982) is good: the secondary vortices in the both lower corners are captured and the asymmetry of the flow pattern is reproduced. A one-to-one comparison is not attempted as the different nature of the fluid (compressible vs. incompressible) inevitably results in differences.

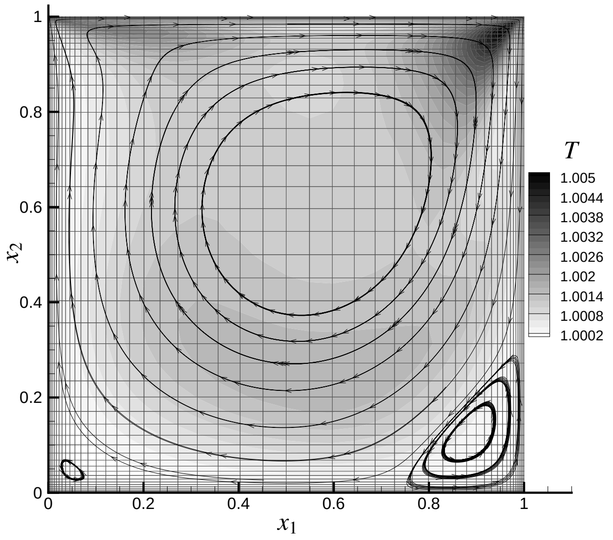


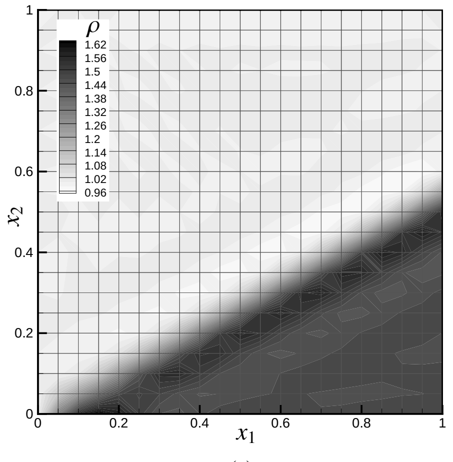
Figure 4.3: Computational mesh and streamlines for the compressible driven cavity flow with $\mathrm{Re}_\eta = 400$ at steady state. The shading indicates the temperature.

4.5.3 Oblique shock

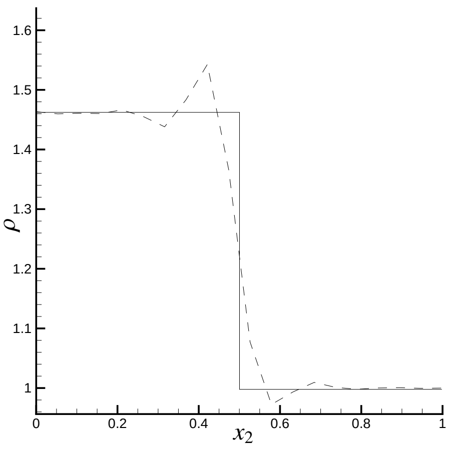
To illustrate the range of applicability of the stabilization matrix, a result for a compressible inviscid test case (also presented by Hauke and Hughes (1998)) is added: an oblique shock forming due to a Mach 2 incident flow at an angle of 10° to a slip wall at the lower boundary of the computational domain. Again, an ideal gas with $\gamma = 7/5$ and $\mathrm{Pr} = 0.715$ is assumed. The Eckert number is $\mathrm{Ec} = (\gamma - 1)M^2 = 1.6$, according to the cited Mach number $M = 2$. The left and top boundary of the unit square are supersonic inflow boundaries with prescribed values for the velocity, $\boldsymbol{v} = \cos(10°)\,\boldsymbol{e}_1 - \sin(10°)\,\boldsymbol{e}_2$, pressure, $p = \rho R T$, and temperature, $T = 1$. At the outflow boundary on the right, no conditions are imposed. The lower boundary is the slip wall, hence the normal component of the velocity is enforced to vanish, $v_2 = 0$. For the other variables there are no conditions. The computations are initialized in the whole domain with the pressure and temperature that are used at the inflow boundary; the initial velocity is set parallel to the wall: $\boldsymbol{v} = \boldsymbol{e}_1$.

Results

The described flow cannot successfully be simulated without stabilization, but the new stabilization operator is successful also in this case. No discontinuity capturing operator is used, which explains the overshoots close to the shock, cf. Figure 4.4 on the next page. For the shown computations, the stabilization parameter has been set to $\omega = 0$. Other choices (e.g., $\omega = \pm 0.4$) are also possible, but the effects on the accuracy of the result or the convergence rate are miniscule.



(a)



(b)

Figure 4.4: Density field for the Mach 2 oblique shock problem. (a) The left and top edges of the domain are inflow boundaries. The right edge is the outflow and along the base only $u_2 = 0$ is enforced. (b) Density field along $x_1 = 0.9$. The solid line is the exact solution with a shock at $x_2 = 0.5$; the dashed line is the numerical solution.

4.6 Conclusions

Based on the unified formulation of the Navier–Stokes equations using entropy or pressure primitive variables, a Galerkin least-squares finite element discretization has been presented that is suitable for both compressible and incompressible flows. A related difficulty is to design a stabilization matrix for the Galerkin least-squares term that is suitable for both types of flow. Such a construction—based on a dimensional analysis of the stabilization matrix for the primitive variables and the important symmetrization property of the entropy variables—is detailed in (Polner, Pesch, and van der Vegt, 2007). The matrix depends on the compressibility parameters of the simulated medium and leads to a formulation whose implementation allows to perform both compressible and incompressible flow computations. Three test cases have been presented here and it has been verified that the stabilization matrix does not degrade the order of accuracy of the method.

These results are positive, but a few disadvantages reduce the practicality of the continuous least-squares method developed so far. These concern mainly the applicability of continuous FEMs in more complicated geometries. The combination of elements with different reference geometry (cf. Chapter 3) requires a matching of the local basis functions across the faces, which may be hard to realize. The same holds for including local refinement:

- p-adaptation, i.e., locally increased polynomial order, again needs a matching of the basis functions across element boundaries.
- Local h-refinement leads to *hanging nodes*, which cannot be accommodated easily in the nodal basis function construction. They have to be removed either geometrically (by appropriate refinement of the neighboring elements) or algebraically (when assembling the system of equations).

On the practical side, ambiguities can occur during the imposition of boundary conditions at points where faces with different conditions meet. In this case, the constraint for the nodal degrees of freedom is not uniquely determined.[2] In general, boundary conditions are cumbersome to implement.

Furthermore, the global basis functions require also a global step in the computation—typically in an assembly procedure for the equation system. If remeshing or refinement is applied then these, too, entail global operations. These are disadvantageous, especially on current parallel computer architectures.

All in all, several disadvantages reduce the flexibility of the method. This is the reason why the quest for a widely applicable method as specified in Chapter 1 has taken a different direction. The discontinuous Galerkin method presented in the following chapter alleviates some of the above problems and combines geometric flexibility with a high degree of locality of the discretization.

[2]This situation occurs, for example, at the two top corners of the driven cavity, where the moving lid and the fixed side walls meet.

Chapter 5

A discontinuous Galerkin finite element method for the Navier–Stokes equations

La continuité dégoûte en tout.

Blaise Pascal (1623–1662), *Pensées*

5.1 Introduction

The continuous Galerkin discretization from Chapter 4 meets some of the goals set out for this work but also has a few inherent disadvantages for the target applications. Some of these can be overcome by a class of methods that will be considered now: discontinuous Galerkin (DG) finite element methods. The main focus of this chapter is the finite element discretization detailed in Section 5.2. The numerical method combines the general variable formulation (with entropy variables constituting a particular choice) with a DG discretization previously used for ideal gas flow (van der Vegt and van der Ven, 2002b; van der Ven and van der Vegt, 2002). The interest in DG methods is partly rooted in their ability to handle (almost) discontinuous solutions, which naturally arise in nonlinear hyperbolic problems, see (Cockburn, 1999; Cockburn et al., 2000) for surveys and various applications. Furthermore, they allow higher order accuracy than traditional (e.g., finite volume) methods for such problems. Compared to traditional (continuous) FEMs, the advantages of this type of discretization include the increased locality both in the sense of data dependence and regarding the possibility of more accurate solution representation. Thereby local hp-adaptation is accommodated naturally, i.e., h-refinement for increasing the mesh resolution in space regions where (possibly discontinuous) small scale flow structures need to be captured (van der Vegt and van der Ven, 2002b; Klaij et al., 2006b), and p-adaptation for representing the solution with higher order smooth basis functions per element (Houston et al., 2006). By using a space-time weak formulation and (discontinuous) finite-element basis functions in space and time, the method can be readily combined with an arbitrary Lagrangian–Eulerian (ALE) formulation (van der Vegt and van der Ven, 2002b) to tackle problems with moving and deforming meshes and boundaries.

Barth (1999) has analyzed the standard DG and Galerkin least-squares variational formulation with entropy variables and has proven (nonlinear) entropy stability for several choices of the numerical flux. The work includes a DG discretization of the Euler equations for ideal gases and several computations of sub- and supersonic flow. In his implementation, Barth uses a Newton iteration for the solution of the system of equations arising from the discretization. While the global linearization of the entropy variable formulation refers back to the well-behaved Jacobians of the quasi-linear form, its implementation is cumbersome and the computation costly. In the algorithm developed here, the linearization is avoided by using a pseudo-time integration method for solving the nonlinear system of equations. This technique augments the (time-dependent) equations with an additional artifical time coordinate, for which the computed solution at a previous time level is the initial condition, and for which a steady state solution is sought. The steady-state in pseudo-time is approached by integrating with Runge–Kutta methods, often adapted to this special purpose (Melson et al., 1993; Kleb et al., 1999; van der Vegt and van der Ven, 2002b; Klaij et al., 2006a,b). The advantage of these methods is that the local character of the DG discretization is—as far as possible—conserved because no global operations or data structures are required. Pseudo-time methods have been analyzed by Klaij et al. (2006a) for the ideal gas Navier–Stokes equations and found beneficial in particular for advection-dominated flow. An important question is, however, how to combine the pseudo-time method with the general variable approach. The analysis by Klaij et al. cannot answer this question as it considers a simplified problem—the scalar advection-diffusion equation—which does not accommodate the transformation that is interposed in the generalized variable formulation. The stability of the solution of the nonlinear system of equations for entropy and pressure primitive variables with a pseudo-time method is analyzed in Section 5.3. The findings presented in this chapter demonstrate how the pseudo-time integration can be combined with the generalized variable approach, what the impact of different fluid models on the algebraic equation system is, and which measures have to be taken when the incompressible limit is attained.

The analysis of the nonlinear solver concerns the Euler equations; the discretization is also extended with the viscous terms of the Navier–Stokes system. For this purpose, the second order partial differential equations are temporarily rewritten as a system of first order equations. For the two sets of first order equations, weak forms are derived. In one of the early applications of a DG FEM to the Navier–Stokes equations, Bassi and Rebay (1997a) solved these equations successively. This approach has the disadvantage that the number of variables is increased.[1] Combining the two weak forms is possible and allows to return to a single variable formulation, as, e.g., exploited in an alternative approach to discretizing the viscous terms by Baumann and Oden (1999). A number of different DG methods have been derived for elliptic problems, see the review by Arnold et al. (2002).

[1]It should also be mentioned that the method by Bassi and Rebay (1997a) suffers from a weak instability, see the discussion in (Arnold et al., 2002).

From these, the interior penalty method (Hartmann, 2006; Hartmann and Houston, 2006) is selected here for the treatment of the viscous terms because it avoids the derivation and costly computation of lifting operators needed, e.g., in (Klaij et al., 2006a,b).

Section 5.4 presents several numerical tests and examples for both compressible and incompressible fluids. This chapter extends the material presented in (Pesch and van der Vegt, 2006, 2007).

5.2 Discontinuous Galerkin discretization

For the purpose of this chapter, the Navier–Stokes equations are initially written in terms of conservation variables U in the form of Eq. (2.41) as the system

$$U_{,t} + F^{\mathsf{e}}_{\bar{p},\bar{p}}(U) = F^{\mathsf{d}}_{\bar{p},\bar{p}}(U, \nabla U) + S \, . \tag{5.1}$$

In Section 2.8.1 it was observed that the viscous flux of the Navier–Stokes equations for Newtonian fluids in the coordinate direction $\bar{\imath}$ is homogeneous of degree one in the conservation variables U, and hence can be expressed as $F^{\mathsf{d}}_{\bar{\imath}} = K^{U}_{\bar{\imath}\bar{n}}(U)\, U_{,\bar{n}}$, with the viscosity matrices $K^{U}_{\bar{\imath}\bar{\jmath}} \in \mathbb{R}^{(d+2)\times(d+2)}$, which depend on the local state and the properties of the fluid. This allows to rearrange the system of second order partial differential equations (5.1) as a set of two systems of first order partial differential equations in the state U and the auxiliary variable θ for the viscous flux. In component form this reads

$$F^{\mathsf{e}}_{\tilde{\imath}p,p} - \theta_{\tilde{\imath}\bar{p},\bar{p}} = S_{\tilde{\imath}} \, , \tag{5.2a}$$

$$\theta_{\tilde{\imath}\bar{\jmath}} = K^{U}_{\bar{\jmath}\bar{r}\tilde{\imath}\tilde{s}}(U)\, U_{\tilde{s},\bar{r}} \, . \tag{5.2b}$$

Again, the time component of the Euler flux, F^{e}_{0}, contains the state variables, U, see (2.15). The viscous fluxes, in contrast, have no component in the time dimension. For the generalized variable approach, (5.2a) remains unchanged except for dependence on the set V instead of the conservation variables U. In Eq. (5.2b), the spatial derivatives of V are premultiplied with the transformed matrices $K^{V}_{\bar{\imath}\bar{\jmath}} = K^{U}_{\bar{\imath}\bar{\jmath}}\, \partial U/\partial V$, hence $\theta_{\tilde{\imath}\bar{\jmath}} = K^{V}_{\bar{\jmath}\bar{r}\tilde{\imath}\tilde{s}}(V)\, V_{\tilde{s},\bar{r}}$.

5.2.1 Function spaces

In this chapter, the functions used to approximate the solution fields are allowed to be discontinuous at element boundaries both in space and time. More specifically, the test and trial spaces for the DG method are based on functions that are elements of the space of tensor product polynomials in space, augmented with monomial basis functions in time, $P_{(p_{\mathrm{t}},p_{\mathrm{s}})}(\hat{\mathcal{K}})$, on the space-time reference element $\hat{\mathcal{K}}$. The maximum order in time and space on the element $\mathcal{K}^{n}_{e}$ is given by $p^{n,e}_{\mathrm{t}}$ and $p^{n,e}_{\mathrm{s}}$, respectively. These properties are reflected in the definition of the function space referring to the n^{th} space time slab,

$$\mathcal{P}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)} := \{w \in L^2(\mathcal{E}_h^n) \mid w|_{\mathcal{K}_e^n} \circ G_e^n \in P_{(p_\mathrm{t}^{n,e},p_\mathrm{s}^{n,e})}(\hat{\mathcal{K}}_e^n)\ \forall\, \mathcal{K}_e^n \in \mathcal{T}^n\}\,. \tag{5.3}$$

The notation $\langle p_\mathrm{t}^{n,e}\rangle_e$ stands for the sequence of all polynomial degrees $p_\mathrm{t}^{n,e}$ on the elements $\mathcal{K}_e^n$ in the n^{th} space-time slab. The space $P_{(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}(\hat{\mathcal{K}}_e^n)$ thus contains the tensor product polynomials of degree $p_\mathrm{s}^{n,e}$ on the spatial reference element $\hat{\mathcal{K}}_e^n$ and of order $p_\mathrm{t}^{n,e}$ in time for this element. Based on the space for scalar functions $\mathcal{P}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}$, the function spaces for state vectors V in the relevant variables, and for the flux and test function matrices X are defined, respectively, as

$$\mathcal{V}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)} := \{V \in (L^2(\mathcal{E}_h^n))^{(2+d)} \mid V_{\tilde{\imath}} \in \mathcal{P}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}\ \forall\, \tilde{\imath} = 1,\dots,2+d\}\,, \tag{5.4}$$

$$\begin{aligned}\mathcal{X}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)} &:= \{X \in (L^2(\mathcal{E}_h^n))^{(2+d)\times d} \mid \\ &\qquad X_{\tilde{\imath}\bar{\jmath}} \in \mathcal{P}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}\ \forall\, \tilde{\imath} = 1,\dots,2+d,\ \bar{\jmath} = 1,\dots,d\}\,.\end{aligned} \tag{5.5}$$

On one element all variables are expanded to the same order. At the internal faces, the polynomial representations of V on the two adjacent elements may lead to different values on the two sides of the face. To deal with these discontinuities, a number of quantities are defined in Section 5.2.2 based on the traces $V^{\pm}_{\mathcal{K}_e^n}$, cf. Definition 3.2 on page 43.

Remark 5.1 *Different order expansions for different variables on the same space-time element are neither necessary nor used here. By contrast, for the continuous Galerkin method used in Chapter 4, using basis functions of one order less for the pressure than for the other components constitutes a possibility to fulfill the LBB criterion for incompressible flow. The least-squares term in Chapter 4 is another means to the same end.* □

5.2.2 Average and jump operators

Definition 5.1 (Averages): *Let $\mathcal{S}$ be an internal or time face connecting the elements $\mathcal{K}_{e_1}^{n_1}$ and $\mathcal{K}_{e_2}^{n_2}$, which may be neighbors within one time slab ($n_1 = n_2 = n$) or across a time slab boundary, and $\mathcal{S}_{e,j}^{\mathrm{b},n} \subset \mathcal{Q}_h$ a boundary face of the element $\mathcal{K}_e^n$. For a function $f \in \mathcal{P}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}$, the* average *of f on a face is defined as*

$$\{\!\{f\}\!\} := \begin{cases} \frac{1}{2}\left(f^-_{\mathcal{K}_{e_1}^{n_1}} + f^-_{\mathcal{K}_{e_2}^{n_2}}\right) & \text{on } \mathcal{S} \in \mathcal{F}^{\mathrm{i},n} \cup \mathcal{F}^{\mathrm{t},n}\,, \\ f^-_{\mathcal{K}_e^n} & \text{on } \mathcal{S}_{e,j}^{\mathrm{b},n} \in \mathcal{F}^{\mathrm{b},n}\,. \end{cases} \tag{5.6}$$

□

Definition 5.2 (Jumps): *Using the same notation as in Definition 5.1, the* jump of $f \in \mathcal{P}_h^{n,(\langle p_\mathsf{t}^{n,e}\rangle_e,\langle p_\mathsf{s}^{n,e}\rangle_e)}$ *is defined as*

$$[\![f]\!] := \begin{cases} f^-_{\mathcal{K}^{n_1}_{e_1}} \boldsymbol{n}^+_{\mathcal{K}^{n_1}_{e_1}} + f^-_{\mathcal{K}^{n_2}_{e_2}} \boldsymbol{n}^+_{\mathcal{K}^{n_2}_{e_2}} & \text{on } \mathcal{S} \in \mathcal{F}^{\mathsf{i},n} \cup \mathcal{F}^{\mathsf{t},n}\,, \\ f^-_{\mathcal{K}^n_e} \boldsymbol{n}^+_{\mathcal{K}^n_e} & \text{on } \mathcal{S}^{\mathsf{b},n}_{e,j} \in \mathcal{F}^{\mathsf{b},n}\,. \end{cases} \tag{5.7}$$

□

Remark 5.2 *1. The two normal vectors $\boldsymbol{n}^+_{\mathcal{K}^{n_1}_{e_1}}$ and $\boldsymbol{n}^+_{\mathcal{K}^{n_2}_{e_2}}$ on the internal face $\mathcal{S}$ are outward with respect to the elements $\mathcal{K}^{n_1}_{e_1}$ and $\mathcal{K}^{n_2}_{e_2}$, respectively, and hence in each point of opposite direction, i.e., $\boldsymbol{n}^+_{\mathcal{K}^{n_1}_{e_1}} = -\boldsymbol{n}^+_{\mathcal{K}^{n_2}_{e_2}}$.*

*2. In the sequel, the notation for the outward normal vector with respect to $\mathcal{K}^n_e$ is simplified as $\boldsymbol{n}^+$. The normal vector $\boldsymbol{n}$ on a face (cf. Definition 3.1) is implied to point from the (arbitrarily assigned) left (*L*) to the right (*R*) element. If the face is a boundary face, then the adjacent element is tagged as left and the normal vector is outward. The limit notation $f^-_{\mathcal{K}^n_e}$ is replaced by f^L and f^R when a term is based on a face integral.* □

In the weak form to be derived, after application of the Gauß theorem to the element integrals, terms of the form

$$\sum_{\mathcal{K}^n_e \in \mathcal{T}^n} \int_{\partial\mathcal{K}^n_e} (W_{\tilde{r}})^-_{\mathcal{K}^n_e} (F_{\tilde{r}p})^-_{\mathcal{K}^n_e}\, n^+_p \,\mathrm{d}(\partial\mathcal{K})\,, \tag{5.8}$$

i.e., the componentwise product of the test function $W_{\tilde{\imath}}$ and the normal flux $F_{\tilde{\imath}p}\, n^+_p$, occur. The range of the summation in (5.8) can be conceptionally changed: In place of summing over the element boundaries, one can rather use the faces $\mathcal{S} \in \mathcal{F}^n$ as summation base. Given the multi-valued state on internal faces, the question is still open how to choose the values of the flux F based on the states U^L and U^R on the left and right sides. This problem is deferred by introducing a *numerical flux function* $\hat{F}$, which assigns a single value to the flux based on the two states. In this way it is clear that the flux is defined uniquely per face and needs to be evaluated only once per face instead of once per element side.[2] Necessary conditions for a conservative and consistent numerical flux function are (i) $\hat{F}(U^\mathsf{L}, U^\mathsf{R}; \boldsymbol{n}) = -\hat{F}(U^\mathsf{R}, U^\mathsf{L}; -\boldsymbol{n})$, and (ii) $\hat{F}(U, U; \boldsymbol{n}) = F(U) \cdot \boldsymbol{n}$.

When considered on a time face, the definition of the numerical flux $\hat{F}_0$ is governed by causality: it has to be upwind in time, cf. Section 5.2.5. With this choice, each space-time slab couples explicitly only to the preceding slab. The limited dependence has been

[2]Obviously the numerical flux function may have to be evaluated several times per face to approximate the integral based on a numerical quadrature rule.

exploited already in the per-time slab definition of the FE spaces in (5.4) and (5.5). By the same token, the summations in the weak form will from now on include only one time slab, and where necessary, previous slab data will be referenced appropriately. The following relation—an application of the more general Equation (3.3) from (Arnold et al., 2002) to (5.8)—is easily verified by inserting Definitions 5.1 and 5.2,

$$
\begin{aligned}
&\sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\int_{\partial\mathcal{K}_e^n}(W_{\tilde r})^-_{\mathcal{K}_e^n}(F_{\tilde r p})^-_{\mathcal{K}_e^n}\,n_p^+\,\mathrm{d}(\partial\mathcal{K})\\
&=\sum_{\mathcal{S}\in\mathcal{F}^{\mathsf{i},n}}\int_{\mathcal{S}}\Big([\![W_{\tilde r}]\!]_p\{\!\{\hat F_{\tilde r p}\}\!\}+\{\!\{W_{\tilde r}\}\!\}\,[\![\hat F_{\tilde r p}]\!]_p\Big)\,\mathrm{d}\mathcal{S}+\sum_{\mathcal{S}\in\mathcal{F}^{\mathsf{b},n}}\int_{\mathcal{S}}(W_{\tilde r})^-_{\mathcal{K}_e^n}F_{\tilde r p}\,n_p\,\mathrm{d}\mathcal{S}\\
&+\sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\left[-\int_{\bar{\mathcal{K}}_e^{n-1,+}}(W_{\tilde r})^-_{\mathcal{K}_e^n}(F_{\tilde r0})^+_{\mathcal{K}_e^n}\,\mathrm{d}\bar{\mathcal{K}}+\int_{\bar{\mathcal{K}}_e^{n,-}}(W_{\tilde r})^-_{\mathcal{K}_e^n}(F_{\tilde r0})^-_{\mathcal{K}_e^n}\,\mathrm{d}\bar{\mathcal{K}}\right].
\end{aligned}
\tag{5.9}
$$

Indices on a jump quantity concern the component of the normal vector, see (5.7). In the boundary integral, the flux has not been marked as interior trace $(\hat F_{\tilde\imath k})^-_{\mathcal{K}_e^n}$ as its determination depends on the boundary condition imposed on the face $\mathcal{S}\in\mathcal{F}^{\mathsf{b},n}$. Several types of boundary conditions are discussed in Section 5.2.7. The superscript '+' on $(F_{\tilde\imath 0})^+_{\mathcal{K}_e^n}$ in the integral over the past time face refers to an external limit, i.e., the data is taken from the previous time slab.[3] For the future time face, the limit is internal, so that $(F_{\tilde\imath 0})^-_{\mathcal{K}_e^n}$ is based on the current time slab, see Section 5.2.5, where also the choice of the spatial numerical flux function for the Euler equations is discussed.

The support of the test function W is a subset of the current time slab only, which simplifies the jump on the time faces. As the numerical method is required to be locally conservative, the jump of the flux on internal faces has to vanish, $[\![\hat F_{\tilde\imath j}]\!]=0$. For a unique flux value, the average becomes $\{\!\{\hat F_{\tilde\imath j}\}\!\}=\hat F_{\tilde\imath j}$. With these replacements and using the definition of the mean and jump on boundary faces, the above relation can be simplified to

$$
\begin{aligned}
\sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\int_{\partial\mathcal{K}_e^n}(W_{\tilde r})^-_{\mathcal{K}_e^n}(F_{\tilde r p})^-_{\mathcal{K}_e^n}\,n_p^+\,\mathrm{d}(\partial\mathcal{K})&=\sum_{\mathcal{S}\in\mathcal{F}^{\mathsf{i},n}\cup\mathcal{F}^{\mathsf{b},n}}\int_{\mathcal{S}}[\![W_{\tilde r}]\!]_p\hat F_{\tilde r p}\,\mathrm{d}\mathcal{S}\\
&+\sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\left[-\int_{\bar{\mathcal{K}}_e^{n-1,+}}(W_{\tilde r})^-_{\mathcal{K}_e^n}(F_{\tilde r0})^+_{\mathcal{K}_e^n}\,\mathrm{d}\bar{\mathcal{K}}+\int_{\bar{\mathcal{K}}_e^{n,-}}(W_{\tilde r})^-_{\mathcal{K}_e^n}(F_{\tilde r0})^-_{\mathcal{K}_e^n}\,\mathrm{d}\bar{\mathcal{K}}\right].
\end{aligned}
\tag{5.10}
$$

[3]The limit could thus also be written as $(F_{\tilde\imath 0})^-_{\mathcal{K}_{e'}^{n-1}}$, where e' is the element index of the element in $\mathcal{E}_h^{n-1}$ that connects to $\mathcal{K}_e^n$ across the face $\mathcal{S}_{e,e'}^{\mathsf{p},n}$.

5.2.3 Weak form of the flux form equations

Equation (5.2a) appears in divergence form and thus lends itself well to the finite element procedure for conservation laws: multiplication with a test function $W \in \mathcal{V}_h^{n,(\langle p_\mathrm{t}^{n,e}\rangle_e,\langle p_\mathrm{s}^{n,e}\rangle_e)}$ and integration over a space-time slab. The latter is represented as the sum of the integrals over space-time elements $\mathcal{K}_e^n$ of the tessellation $\mathcal{T}^n$. Integration by parts yields

$$\sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \left\{ - \int_{\mathcal{K}_e^n} (W_{\tilde{r},p} F^\mathrm{e}_{\tilde{r}p} - W_{\tilde{r},\bar{p}} \theta_{\tilde{r}\bar{p}} + W_{\tilde{r}} S_{\tilde{r}}) \, \mathrm{d}\mathcal{K} + \int_{\partial\mathcal{K}_e^n} (W_{\tilde{r}})^-_{\mathcal{K}_e^n} \left((F^\mathrm{e}_{\tilde{r}p})^-_{\mathcal{K}_e^n} n_p^+ - (\theta_{\tilde{r}\bar{p}})^-_{\mathcal{K}_e^n} n_{\bar{p}}^+ \right) \mathrm{d}(\partial\mathcal{K}) \right\} = 0 \,, \tag{5.11}$$

with $\boldsymbol{n}$ the unit outward normal vector with respect to the element $\mathcal{K}_e^n$. Here it has been exploited that the viscous fluxes in the time direction vanish. That allows to use differentiation only in the spatial directions while integrating over a space-time element. The train of thought of Eq. (5.10) applied to (5.11) yields

$$-\sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \int_{\mathcal{K}_e^n} (W_{\tilde{r},p} F^\mathrm{e}_{\tilde{r}p} - W_{\tilde{r},\bar{p}} \theta_{\tilde{r}\bar{p}} + W_{\tilde{r}} S_{\tilde{r}}) \, \mathrm{d}\mathcal{K} + \sum_{\mathcal{S} \in \mathcal{F}^{\mathrm{i},n} \cup \mathcal{F}^{\mathrm{b},n}} \int_{\mathcal{S}} ([\![W_{\tilde{r}}]\!]_p \hat{F}^\mathrm{e}_{\tilde{r}p} - [\![W_{\tilde{r}}]\!]_{\bar{p}} \hat{\theta}_{\tilde{r}\bar{p}}) \, \mathrm{d}\mathcal{S} + \sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \left[- \int_{\bar{\mathcal{K}}_e^{n-1,+}} (W_{\tilde{r}})^-_{\mathcal{K}_e^n} (U_{\tilde{r}})^+_{\mathcal{K}_e^n} \, \mathrm{d}\bar{\mathcal{K}} + \int_{\bar{\mathcal{K}}_e^{n,-}} (W_{\tilde{r}})^-_{\mathcal{K}_e^n} (U_{\tilde{r}})^-_{\mathcal{K}_e^n} \, \mathrm{d}\bar{\mathcal{K}} \right] = 0 \,. \tag{5.12}$$

The states U on the past and future time faces stem from the time component of the Euler flux, F^e_0, and depend on the state in the discretized variables, viz. $U = U(V)$.

To allow for moving and deforming domains and meshes, van der Vegt and van der Ven (2002b) have shown how—based on the space-time normal vector and fluxes—the above weak form can be interpreted in arbitrary Lagrangian–Eulerian manner: On the space-time faces, the movement of the mesh with the velocity $\boldsymbol{v}^g$ leads to a flux $\hat{U} v^g_{\bar{p}} n_{\bar{p}}$ across the face with space-time normal vector $\boldsymbol{n}$. The ALE form will not be exploited in the sequel, therefore it is not derived and the original reference should be consulted for details.

5.2.4 Weak form of the viscous flux equation

In this section, the homogeneity equation (5.2b) of the first order system formulation of the Navier–Stokes equations is treated. Notably, by rewriting Eq. (5.1) as a system of first order equations (5.2), the number of unknowns that would need to be computed and stored during the solution process is increased significantly. The goal is to return to a single variable formulation by deriving a weak expression for θ in terms of the state variables V and substituting this expression back into (5.12).

The diffusive flux variable θ from Eq. (5.2b) is multiplied with a test function matrix $X \in \mathcal{X}_h^{n,(\langle p_t^{n,e}\rangle_e,\langle p_s^{n,e}\rangle_e)}$ with double contraction and the result is integrated over a physical space element $\bar{\mathcal{K}}_e^n(t)$, viz.,

$$\int_{\bar{\mathcal{K}}_e^n(t)} X_{\tilde{r}\bar{p}}\theta_{\tilde{r}\bar{p}}\,\mathrm{d}\bar{\mathcal{K}} = \int_{\bar{\mathcal{K}}_e^n(t)} X_{\tilde{r}\bar{p}}K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}V_{\tilde{s},\bar{q}}\,\mathrm{d}\bar{\mathcal{K}}\,, \tag{5.13}$$

where $\tilde{r},\tilde{s} = 1,\ldots,2+d$ index the solution components and $\bar{p},\bar{q} = 1,\ldots,d$ concern summation over space dimensions. Partial integration of the term on the right hand side yields

$$\int_{\bar{\mathcal{K}}_e^n(t)} X_{\tilde{r}\bar{p}}K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}V_{\tilde{s},\bar{q}}\,\mathrm{d}\bar{\mathcal{K}} = \int_{\partial\bar{\mathcal{K}}_e^n(t)} (X_{\tilde{r}\bar{p}})^-_{\mathcal{K}_e^n}(K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}})^-_{\mathcal{K}_e^n}\hat{V}_{\tilde{s}}\,n^+_{\bar{q}}\,\mathrm{d}(\partial\bar{\mathcal{K}}) - \int_{\bar{\mathcal{K}}_e^n(t)} (X_{\tilde{r}\bar{p}}K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}})_{,\bar{q}}V_{\tilde{s}}\,\mathrm{d}\bar{\mathcal{K}}\,.$$

The symbol $\hat{V}$ denotes a data vector of values for the variables V representative on the element boundary, i.e., an internal or boundary face, in much the same way as the introduction of numerical fluxes in the conservation equation. How $\hat{V}$ is chosen based on the discontinuous representation on the elements is the topic of Section 5.2.5. The second resultant term is again integrated by parts, this time making use of the fact that the support of the test function is limited to $\mathcal{K}_e^n$, so that the outer limit vanishes,

$$\int_{\bar{\mathcal{K}}_e^n(t)} (X_{\tilde{r}\bar{p}}K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}})_{,\bar{q}}V_{\tilde{s}}\,\mathrm{d}\bar{\mathcal{K}} = \int_{\partial\bar{\mathcal{K}}_e^n(t)} (X_{\tilde{r}\bar{p}})^-_{\mathcal{K}_e^n}(K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}})^-_{\mathcal{K}_e^n}(V_{\tilde{s}})^-_{\mathcal{K}_e^n}\,n^+_{\bar{q}}\,\mathrm{d}(\partial\bar{\mathcal{K}}) - \int_{\bar{\mathcal{K}}_e^n(t)} X_{\tilde{r}\bar{p}}K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}V_{\tilde{s},\bar{q}}\,\mathrm{d}\bar{\mathcal{K}}\,.$$

To coincide with the space-time integral form of Equation (5.2a), the previous equations are integrated over time. As mentioned earlier, there are no diffusive terms in the time direction, so that the time faces can be neglected in the space-time form. With this convention, the integration by parts also includes the time dimension, and summing over all space-time elements in slab $\mathcal{E}_h^n$, the above contributions add up to

$$\begin{aligned}&\sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\int_{\mathcal{K}_e^n} X_{\tilde{r}\bar{p}}\theta_{\tilde{r}\bar{p}}\,\mathrm{d}\mathcal{K}\\ &= \sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\int_{\mathcal{K}_e^n} X_{\tilde{r}\bar{p}}K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}V_{\tilde{s},\bar{q}}\,\mathrm{d}\mathcal{K} + \sum_{\mathcal{K}_e^n\in\mathcal{T}^n}\int_{\partial\mathcal{K}_e^n} (X_{\tilde{r}\bar{p}})^-_{\mathcal{K}_e^n}(K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}})^-_{\mathcal{K}_e^n}(\hat{V}_{\tilde{s}} - (V_{\tilde{s}})^-_{\mathcal{K}_e^n})\,n^+_{\bar{q}}\,\mathrm{d}(\partial\mathcal{K})\,,\end{aligned}$$

to which (5.9) can be applied, so that

$$\begin{aligned} &= \sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \int_{\mathcal{K}_e^n} X_{\tilde{r}\bar{p}} K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}} V_{\tilde{s},\bar{q}} \, \mathrm{d}\mathcal{K} + \sum_{\mathcal{S} \in \mathcal{F}^{\mathsf{i},n}} \int_{\mathcal{S}} [\![X_{\tilde{r}\bar{p}} K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}]\!]_{\bar{q}} \, \{\!\{ \hat{V}_{\tilde{s}} - V_{\tilde{s}} \}\!\} \, \mathrm{d}\mathcal{S} \\ &\quad + \sum_{\mathcal{S} \in \mathcal{F}^{\mathsf{i},n} \cup \mathcal{F}^{\mathsf{b},n}} \int_{\mathcal{S}} \{\!\{ X_{\tilde{r}\bar{p}} K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}} \}\!\} \, [\![\hat{V}_{\tilde{s}} - V_{\tilde{s}}]\!]_q \, \mathrm{d}\mathcal{S} \,. \end{aligned} \tag{5.14}$$

At this stage, the numerical fluxes that occur in Eqs. (5.12) and (5.14) have to be specified to complete the weak formulation.

5.2.5 Choice of the numerical fluxes

In the previous sections, face integrals of various quantities f have occurred and typically the decision how to choose a value representative of the possibly multi-valued function f on the face was deferred by introducing an approximation $\hat{f}$, often called *numerical flux*. It is now time to define these fluxes in terms of the data available from the element(s) neighboring the face.

Time flux

Elements in different time slabs are connected by the time faces. An element $\mathcal{K}_e^n$ in the slab $\mathcal{E}_h^n$ connects to the element $\mathcal{K}_{e'}^{n-1}$ in the previous slab through the face $\mathcal{S}_{e,e'}^{\mathsf{p},n}$ and to the element $\mathcal{K}_{e'}^{n+1}$ through $\mathcal{S}_{e,e'}^{\mathsf{f},n}$.[4] On these faces, the choice of the numerical flux in Eq. (5.12) is based on a causality argument: the current state can only be determined by the past, not by the future; vice versa, the current state can only have an effect on the future, not on the past. Hence the fluxes across time faces are defined as

$$\hat{V} := \begin{cases} V_{e'}^{n-1}\big|_{\mathcal{K}_e^{n-1,-}} & \text{on } \mathcal{S}_{e,e'}^{\mathsf{p},n} \,, \\ V_e^n\big|_{\mathcal{K}_e^{n,-}} & \text{on } \mathcal{S}_{e,e'}^{\mathsf{f},n} \,. \end{cases} \tag{5.15}$$

Euler flux

In Equation (5.12), the integrals of the (numerical) Euler flux $\hat{F}^{\mathsf{e}}$ on the faces are an addition that distinguishes the DG discretization from the method applied in Chapter 4.[5] Because of the discontinuous basis functions, the state on two adjacent elements (or the element and the boundary value) may be different when evaluated on the element boundary.

[4]Per element $\mathcal{K}_e^n$ more than one such face (of either type) can exist on the hyperplanes $\mathcal{H}^{n-1}$ and $\mathcal{H}^n$, if the element is refined or de-refined between time slabs.

[5]It may be argued that this term is also present in the continuous Galerkin weak form, just there all jump quantities vanish so that there is no contribution, except on boundary faces.

To give a meaning to the multivaluedness, an interpretation typically used in finite volume methods for gas dynamics may be followed: The two states are considered the data of a (hypothetic) initial value problem with discontinuous data, a so-called Riemann problem. The solution of the Riemann problem requires additional knowledge about the considered medium and—depending on the amount of detail invested in the solution—different compromises between accuracy and computational cost are possible, cf. (Menikoff and Plohr, 1989; Toro, 1999).

Alternatively, a numerical flux may be defined without the interpretation as a Riemann problem and explicit reference to the medium, making it more generic, possibly at the cost of the accuracy of the description. Remarkably, some seemingly obvious choices are not suitable, for instance the mean flux $\hat{F}(U^{\mathsf{L}}, U^{\mathsf{R}}) = 1/2\,(F(U^{\mathsf{L}}) + F(U^{\mathsf{R}}))$ is unstable, cf. (LeVeque, 2002).

Two choices for the numerical Euler flux function $\hat{F}^{\mathsf{e}}$ that are suitable for the DG method are presented next, along with a discussion of their suitability for the goals set out for the current work. Regarding the notation, the numerical flux functions for the Euler flux are implied to return a normal flux to the face. For this reason, the function is dependent on the normal vector $\boldsymbol{n}$ and constitutes a $(d+2)$-component data vector, $\hat{F}^{\mathsf{e}}_{\bar{\imath}}$. The argument of the space-time face integral in Eq. (5.12) will be adapted so that the normal vector multiplication between the jump of the test function and the Euler flux matrix is moved inside the numerical flux function.

HLLC The HLLC numerical flux for the Euler equations, cf. (Toro et al., 1994; Toro, 1999) for its derivation, is an approximate Riemann solver originally developed in the context of Godunov finite volume methods. It has been applied many times in DG FEMs, e.g. by van der Ven and van der Vegt (2002); Klaij et al. (2006b), and van der Vegt and van der Ven (2002b), who also detail the computation of the flux in space-time.

The HLLC flux is a good choice for computing ideal gas flow, and can be applied to certain real gases (see the applications with the covolume EOS in (Toro et al., 1994) and in Section 5.4.3 of this thesis), but ultimately its derivation restricts it to gaseous media. This restriction applies to many numerical fluxes, which also often do not scale correctly in the low Mach-number/incompressible limit (Guillard and Viozat, 1999; Guillard and Murrone, 2004). This can manifest itself in the deterioration of numerical accuracy and convergence. Consequently such flux functions are not suitable for a numerical method that aims at being applicable for different media and flow conditions.

Because the HLLC flux does not fulfill the requirement of genericity for the development of a unified numerical method for compressible and incompressible flows, a different solution had to be found. An examination of the discontinuous Galerkin FEM with a stabilization operator that includes least-squares contributions from both elements and faces (Houston et al., 2002) seemed promising. It offered the worthwhile effect that the work on least-squares operators for both compressible and incompressible flows (cf. Chapter 4) could have been reused. Houston et al. add a diffusion-like term based on the state difference on the element boundaries. However, for several reasons the investigations in this method were concluded negatively: The available analysis considers linear hyperbolic systems only and would need to be extended. Also, the splitting of the flux Jacobians on the faces is non-trivial, because they concern the variable set V. Even if this could be accomplished analytically, the evaluation of the whole stabilization term is still expected to be computationally expensive and already results obtained for the advection equation were not convincing when related to the computational cost. Finally, the *element* stabilization operator from Chapter 4 could be reused, but whether it could be readily adapted to serve also as the *face* stabilization operator would have to be verified.

For these reasons, the general simulation framework uses the local Lax–Friedrichs flux, which applies to all fluids since it does not exploit specific information, for example about the wave structure. Also in favor of this decision counts the decreasing importance of the specific choice of the numerical flux for higher order DG methods (Persson and Peraire, 2006; van der Vegt, 2006).

(L)LF The Lax–Friedrichs (LF) flux for conservation variables in the direction of a normal $\boldsymbol{n}$ is given by

$$\hat{F}_{\bar{\imath}}^{\mathsf{LF}}(U^{\mathsf{L}}, U^{\mathsf{R}}; \boldsymbol{n}) := \frac{1}{2}\left(F_{\bar{\imath}p}(U^{\mathsf{L}}) + F_{\bar{\imath}p}(U^{\mathsf{R}})\right) n_p - \frac{1}{2}\lambda_{\max}(U_\star; \boldsymbol{n})\left(U_{\bar{\imath}}^{\mathsf{R}} - U_{\bar{\imath}}^{\mathsf{L}}\right), \tag{5.16}$$

with $\lambda_{max}(U_\star; \boldsymbol{n})$ an estimate of the maximum eigenvalue of $A^U(U; \boldsymbol{n}) = n_{\bar{s}}\, \partial F_{\bar{s}} / \partial U$ for the states U on a face. The computation is facilitated by the knowledge of the eigenvalues of the Euler flux Jacobians for conservation variables, cf. (Toro, 1999).

The LF flux adds a diffusion-like term to the (unstable) mean flux to damp numerical instabilities. It is actually overly diffusive, but thanks to its very general nature and independence of the EOS it is often preferred when (mixtures of) several fluids are treated. For the local version (LLF), the eigenvalue computation is based on the mean states on the adjacent element(s) and hence needs to be done only once per element. The global version maximizes the eigenvalue non-locally, which is computationally more involved. Cockburn and Shu (1998) report that they found the local Lax–Friedrichs flux to be well-suited for their Runge–Kutta-DG method, leading to less dissipation, but also decreased robustness, compared to the global Lax–Friedrichs flux.

For entropy variables, Barth (1999) defines the flux as

$$\hat{F}_{\bar{\imath}}^{\mathsf{LF}}(V^{\mathsf{L}}, V^{\mathsf{R}}; \boldsymbol{n}) := \frac{1}{2}\left(F_{\bar{\imath}p}(V^{\mathsf{L}}) + F_{\bar{\imath}p}(V^{\mathsf{R}})\right) n_p - \frac{1}{2}\lambda_{\max}(V_\star, \boldsymbol{n})\,(A_0^V)_{\bar{\imath}\bar{r}}\left(V_{\bar{r}}^{\mathsf{R}} - V_{\bar{r}}^{\mathsf{L}}\right), \tag{5.17}$$

and proves nonlinear entropy stability. The matrix $A_0^V = A_0^V(V_\star)$ is evaluated with the state for which the eigenvalue of $A^U(V_\star, \boldsymbol{n})$ is maximized, $\lambda_{\max}(V_\star, \boldsymbol{n})$, which is usually determined with a one- or two-point approximation in state space.

$\hat{V}$ on space-time faces and the viscous flux $\hat{\theta}$

The development of discontinuous Galerkin methods for elliptic problems furnished a multitude of numerical fluxes for such equations. An overview is given by Arnold et al. (2002). For the current work only one specific choice will be used and hence described: the interior penalty (IP) method (Hartmann, 2006; Hartmann and Houston, 2006). For this method, the numerical flux $\hat{V}$ is defined as

$$\hat{V}_{\tilde{\imath}} := \begin{cases} \{\!\{V_{\tilde{\imath}}\}\!\} & \text{on } \mathcal{S}^{\mathsf{i},n}_{\{e_1,e_2\}}\,, \\ V^{\mathsf{b}}_{\tilde{\imath}}(V^{\mathsf{L}}) & \text{on } \mathcal{S}^{\mathsf{b},n}_{e,j}\,. \end{cases} \tag{5.18}$$

with the boundary data $V^{\mathsf{b}}_{\tilde{\imath}}$, which accommodates, for example, Dirichlet (a value $V^{\mathsf{b}}_{\tilde{\imath}}$ is specified) and Neumann ($V^{\mathsf{b}}_{\tilde{\imath}} = V^{\mathsf{L}}_{\tilde{\imath}}$) conditions. The numerical viscous flux is chosen as

$$\begin{array}{rl} \hat{\theta}_{\tilde{\imath}\tilde{\jmath}} := \{\!\{F^{\mathsf{d}}_{\tilde{\imath}\tilde{\jmath}}\}\!\} - \delta_{\tilde{\imath}}\, \{\!\{(A^{V}_{0})_{\tilde{\imath}\tilde{s}}\}\!\} [\![V_{\tilde{s}}]\!]_{\tilde{\jmath}} & \text{on } \mathcal{S}^{\mathsf{i},n}_{\{e_1,e_2\}}\,, \\ \left.\begin{array}{r} \hat{\theta}_{\tilde{\imath}\tilde{p}}\, n_{\tilde{p}} := b^{\mathsf{N}}_{\tilde{\imath}} \\ \hat{\theta}_{\tilde{\imath}\tilde{\jmath}} := F^{\mathsf{d}}_{\tilde{\imath}\tilde{\jmath}}(V^{\mathsf{L}}) - \delta_{\tilde{\imath}}\, (A^{V}_{0})_{\tilde{\imath}\tilde{s}}\, (V^{\mathsf{L}} - V^{\mathsf{b}}(V^{\mathsf{L}}))_{\tilde{s}}\, n_{\tilde{\jmath}} \end{array}\right\} & \text{on } \mathcal{S}^{\mathsf{b},n}_{e,j} \begin{cases} \text{with Neumann b.c.,} \\ \text{with Dirichlet b.c.,} \end{cases} \end{array} \tag{5.19}$$

with Neumann data $b^{\mathsf{N}}_{\tilde{\imath}}$. Here, again, the transformation matrix A^{V}_{0} has been interposed to match the units of the two summands. The penalization parameter $\delta_{\tilde{\imath}} = \delta_{\tilde{\imath}}(\mathcal{S})$ on a face $\mathcal{S}$ that connects the elements $\mathcal{K}^{n}_{e}$ and $\mathcal{K}^{n}_{e'}$ is given by Hartmann and Houston (2006) as

$$\delta_{\tilde{\imath}}(\mathcal{S}) = C_{\mathsf{IP}} \frac{\eta\, (p^{\mathsf{max}}_{\mathsf{s}})^2}{h}\,, \tag{5.20}$$

where $h = \min\{\mathrm{meas}(\mathcal{K}^{n}_{e}), \mathrm{meas}(\mathcal{K}^{n}_{e'})\}/\,\mathrm{meas}(\mathcal{S})$ estimates the spatial element extent in the direction orthogonal to the face $\mathcal{S}$, $p^{\mathsf{max}}_{\mathsf{s}}$ is the maximum polynomial degree on the adjacent element(s), η the dynamic viscosity, and C_{IP} a sufficiently large positive constant stabilization parameter, cf. (Hartmann and Houston, 2006) for its choice. For boundary faces with other than Neumann conditions the same definition of $\delta_{\tilde{\imath}}$ with only one connecting element applies. If a Neumann condition is applied for the $\tilde{\imath}^{\text{th}}$ component, then no penalty applies and $\delta_{\tilde{\imath}}(\mathcal{S}) = 0$.[6] For all numerical examples in Section 5.4, the value $C_{\mathsf{IP}} = 10$ is chosen and the same parameter $\delta_{\tilde{\imath}}$ is used for all components of the system, so that the component index $\tilde{\imath}$ will be dropped from now on.

5.2.6 Primal form of the weak formulation

Given the definition of the numerical viscous fluxes, the derivation of a weak form involving only the unknowns V continues. To replace the element integral of the diffusive flux in Eq. (5.12), the test function in (5.14) is chosen as $X_{\tilde{\imath}\tilde{\jmath}} = W_{\tilde{\imath},\tilde{\jmath}}$. Inserting the viscous numeri-

[6] With this definition, splitting up the boundary face terms according to the condition applied per component is avoided.

cal fluxes, using the additivity and idempotence of the average operator and observing that on internal faces $[\![\hat{V}]\!] = [\![\{\!\{V\}\!\}]\!] = 0$, the primal formulation of the Navier–Stokes equations with interior penalty is:

Find $V \in \mathcal{V}_h^{n,(\langle p_{\mathsf{t}}^{n,e}\rangle_e,\langle p_{\mathsf{s}}^{n,e}\rangle_e)}$ such that for all $W \in \mathcal{V}_h^{n,(\langle p_{\mathsf{t}}^{n,e}\rangle_e,\langle p_{\mathsf{s}}^{n,e}\rangle_e)}$ holds

$$
\begin{aligned}
&- \sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \int_{\mathcal{K}_e^n} (W_{\tilde{r},p} F^{\mathsf{e}}_{\tilde{r}p} - W_{\tilde{r},\bar{p}} F^{\mathsf{d}}_{\tilde{r}\bar{p}} + W_{\tilde{r}} S_{\tilde{r}})\,\mathrm{d}\mathcal{K} \\
&+ \sum_{\mathcal{S} \in \mathcal{F}^{\mathsf{i},n}} \int_{\mathcal{S}} \Big\{ (W^{\mathsf{L}}_{\tilde{r}} - W^{\mathsf{R}}_{\tilde{r}}) \hat{F}^{\mathsf{e}}_{\tilde{r}} - [\![W_{\tilde{r}}]\!]_{\bar{p}} \Big(\{\!\{F^{\mathsf{d}}_{\tilde{r}\bar{p}}\}\!\} - \delta(\mathcal{S})\, \{\!\{(A_0^V)_{\tilde{r}\tilde{s}}\}\!\} [\![V_{\tilde{s}}]\!]_{\bar{p}} \Big) \Big\}\,\mathrm{d}\mathcal{S} \\
&+ \sum_{\mathcal{S} \in \mathcal{F}^{\mathsf{b},n}} \int_{\mathcal{S}} \Big\{ W^{\mathsf{L}}_{\tilde{r}} \hat{F}^{\mathsf{e}}_{\tilde{r}}(V^{\mathsf{L}}, V^{\mathsf{b}}(V^{\mathsf{L}}); \boldsymbol{n}) \\
&\qquad\qquad - W^{\mathsf{L}}_{\tilde{r}} n_{\bar{p}} \Big(F^{\mathsf{d}}_{\tilde{r}\bar{p}}(V^{\mathsf{L}}) - \delta(\mathcal{S})\, (A_0^V)_{\tilde{r}\tilde{s}}\, (V^{\mathsf{L}} - V^{\mathsf{b}}(V^{\mathsf{L}}))_{\tilde{s}}\, n_{\bar{p}} \Big) \Big\}\,\mathrm{d}\mathcal{S} \\
&- \sum_{\mathcal{S} \in \mathcal{F}^{\mathsf{i},n}} \int_{\mathcal{S}} \{\!\{W_{\tilde{r},\bar{p}} K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}\}\!\}\, [\![V_{\tilde{s}}]\!]_{\bar{q}}\,\mathrm{d}\mathcal{S} - \sum_{\mathcal{S} \in \mathcal{F}^{\mathsf{b},n}} \int_{\mathcal{S}} W_{\tilde{r},\bar{p}} K^V_{\bar{p}\bar{q}\tilde{r}\tilde{s}}\, (V^{\mathsf{L}}_{\tilde{s}} - V^{\mathsf{b}}_{\tilde{s}}(V^{\mathsf{L}}))\, n_{\bar{q}}\,\mathrm{d}\mathcal{S} \\
&- \sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \int_{\bar{\mathcal{K}}_e^{n-1,+}} (W_{\tilde{r}})^-_{\mathcal{K}_e^n} U_{\tilde{r}}(V^+_{\mathcal{K}_e^n})\,\mathrm{d}\bar{\mathcal{K}} + \sum_{\mathcal{K}_e^n \in \mathcal{T}^n} \int_{\bar{\mathcal{K}}_e^{n,-}} (W_{\tilde{r}})^-_{\mathcal{K}_e^n} U_{\tilde{r}}(V^-_{\mathcal{K}_e^n})\,\mathrm{d}\bar{\mathcal{K}} = 0\,.
\end{aligned}
\tag{5.21}
$$

As mentioned above, the jump of the test function in the face integral of the numerical Euler flux has been rewritten as the difference of the test function values on the two sides of an internal face because the numerical flux typically immediately returns a normal flux. In the boundary face integrals, the numerical fluxes are based on the internal state V^{L} and a boundary state $V^{\mathsf{b}}(V^{\mathsf{L}})$, which may also depend on the internal state. Examples of such boundary states and the resulting fluxes are given next.

5.2.7 Boundary conditions

At the boundary Q_h, boundary conditions have to be imposed according to the physical conditions. Different scenarios are inflow and outflow, and different interfacial conditions like slip surfaces (along which the fluid may move tangentially) or no-slip ones (where the velocity difference between fluid and boundary has to vanish). The determination which set of quantities has to be prescribed to obtain a well-defined formulation is a topic of profound research. Apart from the type of the boundary, also the properties of the fluid play a significant role: here another difference between the treatment of compressible and incompressible fluids surfaces. In contrast to other instances of such discrepancies, with boundary conditions the dependence on the fluid cannot be circumvented. Hence, in this section, different conditions are specified for compressible and incompressible cases.

Notably, some of the compressible type boundary conditions are derived for an ideal gas again, and as such should be applied with care to other fluids.

In the DG context, some types of boundary conditions can be implemented by transforming the prescribed values to the currently used variable set and passing these values as the external state to the numerical flux. This simplifies the imposition of the condition compared to continuous Galerkin methods (see Chapter 4), where boundary conditions have to be imposed in the global set of equations. In the DG implementation, even if the numerical flux cannot be used on the boundary, the boundary condition specification is still local. Possible conditions for the computational framework are:

Subsonic in-/outflow

A commonly used condition for compressible subsonic inviscid flow is to specify entropy s and stagnation enthalpy $h + k = e + p\alpha + k$ at the inflow and pressure at the outflow boundary. To obtain a complete set of variables, at the inflow the flow angle(s) must also be prescribed and the internal pressure is used for the determination of the external state, too. At the outflow, a set of internal values is required to construct a state with the prescribed pressure.

Darmofal et al. (2000) exposed the exponential decay of initial disturbances when this set of boundary conditions is used, though the decay rate tends to zero for low Mach numbers. In practice, especially the amount of thermodynamical relations that are needed for the construction of the external state from the given values for this type of boundary conditions is large, so that both the implementation and computation is costly.

In-/outflow far-field state

In some instances, e.g. the computation of very low Mach number flow, cf. Section 5.4.1, rather than the previous pair of in- and outflow conditions, the external states of the numerical flux are prescribed completely from given values, independent of the internal flow state. This choice is simple to implement and, especially for steady-state cases, often leads to indistinguishable results compared to the previous choice.

Incompressible in-/outflow

For incompressible fluids, at the inflow the value of the pressure is taken from the internal state, the remaining values are prescribed. At outflow boundaries, the normal stress is prescribed (for inviscid flow or high Reynolds number this reduces to the pressure) and the tangential stress should vanish. See also Section 4.3 and (Wesseling, 2000).

Slip boundary

For the simulation of the limit of inviscid flow with the Euler equations, the condition at a material boundary is that the flow is tangential to the boundary. This can be enforced in different ways. One possibility (van der Vegt and van der Ven, 2002a) is to prescribe the external state of the numerical flux in such a way that the evaluation of the numerical flux will lead to tangential flow. Obviously, the determination of such an external state requires knowledge of the internals of the flux evaluation and may be different for a different numerical flux function. This restriction, together with the fact that the HLLC flux used by van der Vegt and van der Ven is defined for compressible fluids only, made it necessary to look for an alternative way to evaluate the boundary flux. By determining the tangential component of the velocity and using it, together with the internal pressure, for the computation of the exact spatial Euler flux (2.15), the slip flow computation is general. Apart from the variables stipulated so far, one more quantity remains to be specified, typically temperature. Its value can be chosen to obtain different effects: Either a prescribed temperature value can be imposed as Dirichlet condition, or, to simulate an adiabatic wall (not permitting heat flow, $\boldsymbol{n} \cdot \nabla T = 0$, i.e., a homogeneous Neumann condition), the internal temperature is used.

No-slip boundary

When the fluid is attributed a finite viscosity, a boundary layer will form at a material surface due to friction. The effect is modeled by assuming a no-slip condition at the surface. As with the slip boundary, if one wants to stay independent of the numerical flux, the enforcement of the no-slip condition should be based on the exact Euler flux. To evaluate the flux, the internal pressure is used together with the prescribed velocity of the boundary. For temperature, the same possibilities exist as in the slip boundary flux.

5.2.8 The nonlinear set of equations and its solution

Based on the weak form (5.21), a set of equations is obtained by choosing the basis and test functions. The basis functions $\hat{\psi}$ are used directly as test functions and the basis is defined as tensor product of the monomials with the aforementioned order $p_{\mathrm{s}}^{n,e}$ in space combined with the monomials up to order $p_{\mathrm{t}}^{n,e}$ in time. In two-dimensional space, for example, the following sequence of polynomials in reference coordinates $\hat{x}_i$ is chosen as a basis of $P_{(\langle p_{\mathrm{t}}^{n,e}\rangle_e,\langle p_{\mathrm{s}}^{n,e}\rangle_e)}(\hat{\mathcal{K}})$: $[\hat{\psi}_0 = 1,\ \hat{\psi}_1 = \hat{x}_1,\ \hat{\psi}_2 = \hat{x}_2,\ \hat{\psi}_3 = \hat{x}_1\hat{x}_2,\ \hat{\psi}_4 = \hat{x}_0]$, where it has been assumed that the reference time $\hat{x}_0$ maps to $x_0 = t$. Limiting the number of basis functions to 1, 4, or 5, respectively, includes the standard cases (i) constant representation per element ($p_{\mathrm{t}}^{n,e} = p_{\mathrm{s}}^{n,e} = 0$), resulting in a first order convergent numerical method, (ii) constant in time and bilinear in space ($p_{\mathrm{t}}^{n,e} = 0,\ p_{\mathrm{s}}^{n,e} = 1$), which makes the spatial discretization second order accurate, and (iii) second order description in both space and time ($p_{\mathrm{t}}^{n,e} = 1,\ p_{\mathrm{s}}^{n,e} = 1$). Corresponding bases for other dimensions and the generalization to higher order are straightforward to procure.

The basis functions exist on each element and hence the physical space basis functions are enumerated as $\psi^n_{(e,\tilde{k},f)}$, where e represents the element number in slab n, $\tilde{k} \in \{1,\dots,d{+}2\}$ gives the equation number, and f indexes the local basis functions per element. For a variable vector $V \in \mathcal{P}_h^{n,(\langle p_{\mathrm{t}}^{n,e}\rangle_e,\langle p_{\mathrm{s}}^{n,e}\rangle_e)}$ the expansion coefficient matrix on element $\mathcal{K}_e^n$ is denoted as $\check{V}_e^n$ and all expansion coefficients related to the n th space-time slab are subsumed as $\check{V}^n$.

For $V \in \mathcal{V}_h^{n,(\langle p_{\mathrm{t}}^{n,e}\rangle_e,\langle p_{\mathrm{s}}^{n,e}\rangle_e)}$, the terms from (5.21) lead to the definitions (no summation on double index $\tilde{k}$),

$$A^n_{(e,\tilde{k},f)}(V) := -\int_{\mathcal{K}_e^n} \left(\frac{\partial \psi^n_{(e,\tilde{k},f)}}{\partial x_p} F^{\mathrm{e}}_{\tilde{k}p}(V) + \psi^n_{(e,\tilde{k},f)} S_{\tilde{k}} \right) \mathrm{d}\mathcal{K}\,, \tag{5.22a}$$

$$B^n_{(e,\tilde{k},f)}(V) := -\int_{\tilde{\mathcal{K}}_e^{n-1,+}} (\psi^n_{(e,\tilde{k},f)})^-_{\mathcal{K}_e^n}\, U_{\tilde{k}}(V^+_{\mathcal{K}_e^n})\, \mathrm{d}\tilde{\mathcal{K}} + \int_{\tilde{\mathcal{K}}_e^{n,-}} (\psi^n_{(e,\tilde{k},f)})^-_{\mathcal{K}_e^n}\, U_{\tilde{k}}(V^-_{\mathcal{K}_e^n})\, \mathrm{d}\tilde{\mathcal{K}}\,, \tag{5.22b}$$

$$\begin{aligned} C^n_{(e,\tilde{k},f)}(V) := &\sum_{\mathcal{S}\in\mathcal{F}^{\mathrm{i},n}} \int_{\mathcal{S}} (\psi^{n,\mathrm{L}}_{(e,\tilde{k},f)} - \psi^{n,\mathrm{R}}_{(e,\tilde{k},f)})\, \hat{F}^{\mathrm{e}}_{\tilde{k}}(V^{\mathrm{L}}, V^{\mathrm{R}}; \boldsymbol{n})\, \mathrm{d}\mathcal{S} \\ &+ \sum_{\mathcal{S}\in\mathcal{F}^{\mathrm{b},n}} \int_{\mathcal{S}} \psi^{n,\mathrm{L}}_{(e,\tilde{k},f)}\, \hat{F}^{\mathrm{e}}_{\tilde{k}}(V^{\mathrm{L}}, V^{\mathrm{b}}(V^{\mathrm{L}}); \boldsymbol{n})\, \mathrm{d}\mathcal{S}\,, \end{aligned} \tag{5.22c}$$

$$D^n_{(e,\tilde{k},f)}(V) := \int_{\mathcal{K}_e^n} \frac{\partial \psi^n_{(e,\tilde{k},f)}}{\partial x_{\bar{p}}} K^V_{\bar{p}\bar{q}\tilde{k}\tilde{s}} V_{\tilde{s},\bar{q}}\, \mathrm{d}\mathcal{K}\,, \tag{5.22d}$$

$$\begin{aligned} E^n_{(e,\tilde{k},f)}(V) := &-\sum_{\mathcal{S}\in\mathcal{F}^{\mathrm{i},n}} \int_{\mathcal{S}} [\![\psi^n_{(e,\tilde{k},f)}]\!]_{\bar{p}} \left(\{\!\{K^V_{\bar{p}\bar{q}\tilde{k}\tilde{s}} V_{\tilde{s},\bar{q}}\}\!\} - \delta(\mathcal{S})\, \{\!\{(A_0^V)_{\tilde{k}\tilde{s}}\}\!\} [\![V_{\tilde{s}}]\!]_{\bar{p}} \right) \mathrm{d}\mathcal{S} \\ &-\sum_{\mathcal{S}\in\mathcal{F}^{\mathrm{b},n}} \int_{\mathcal{S}} \psi^{n,\mathrm{L}}_{(e,\tilde{k},f)}\, n_{\bar{p}} \left(K^V_{\bar{p}\bar{q}\tilde{k}\tilde{s}} V^{\mathrm{L}}_{\tilde{s},\bar{q}} - \delta(\mathcal{S})\, (A_0^V)_{\tilde{k}\tilde{s}} (V^{\mathrm{L}}_{\tilde{s}} - V^{\mathrm{b}}_{\tilde{s}}(V))\, n_{\bar{p}} \right) \mathrm{d}\mathcal{S}\,, \end{aligned} \tag{5.22e}$$

$$\begin{aligned} F^n_{(e,\tilde{k},f)}(V) := &-\sum_{\mathcal{S}\in\mathcal{F}^{\mathrm{i},n}} \int_{\mathcal{S}} \{\!\{ \frac{\partial \psi^n_{(e,\tilde{k},f)}}{\partial x_{\bar{p}}} K^V_{\bar{p}\bar{q}\tilde{k}\tilde{s}} \}\!\} [\![V_{\tilde{s}}]\!]_{\bar{q}}\, \mathrm{d}\mathcal{S} \\ &-\sum_{\mathcal{S}\in\cup\mathcal{F}^{\mathrm{b},n}} \int_{\mathcal{S}} \frac{\partial \psi^n_{(e,\tilde{k},f)}}{\partial x_{\bar{p}}} K^V_{\bar{p}\bar{q}\tilde{k}\tilde{s}} (V^{\mathrm{L}}_{\tilde{s}} - V^{\mathrm{b}}_{\tilde{s}}(V^{\mathrm{L}}))\, n_{\bar{q}}\, \mathrm{d}\mathcal{S}\,. \end{aligned} \tag{5.22f}$$

In the definition of $A^n_{(e,\tilde{k},f)}$, $B^n_{(e,\tilde{k},f)}$, and $D^n_{(e,\tilde{k},f)}$ one can immediately make use of the fact that the support of $\psi^n_{(e,\tilde{k},f)}$ is limited to a single element $\mathcal{K}_e^n$. In the same way it is clear that $C^n_{(e,\tilde{k},f)}$, $E^n_{(e,\tilde{k},f)}$, and $F^n_{(e,\tilde{k},f)}$ only yield contributions from faces $\mathcal{S}$ that neighbor $\mathcal{K}^n_e$. For a single choice of the test function W as $\psi^n_{(e,\tilde{k},f)}$, the inviscid (Euler) part of Eq. (5.12) becomes

$$\mathcal{L}^{\mathrm{e},n}_{(e,\tilde{k},f)}(\check{V}^n;\check{V}^{n-1}) := A^n_{(e,\tilde{k},f)}(\check{V}^n) + B^n_{(e,\tilde{k},f)}(\check{V}^n;\check{V}^{n-1}) + C^n_{(e,\tilde{k},f)}(\check{V}^n) = 0\,, \tag{5.23}$$

and the set of all these equations for the n^{th} space-time slab is subsumed as

$$\mathcal{L}^{\mathrm{e},n}(\check{V}^n;\check{V}^{n-1}) = 0\,, \tag{5.24}$$

including the unknowns $\check{V}^n$ and the known solution $\check{V}^{n-1}$ from the previous space-time slab, which enters through the integral over the 'past' time faces in (5.22b). Note that the method developed in this article is implicit, no Courant–Friedrichs–Lewy (CFL) criterion applies, and only the accuracy requirement limits the time step. The evaluation of all integrals in (5.22) is based on Gauß quadrature rules of order $2(p+1)$ on the faces and $2p+1$ on the elements, with p the maximum order of the polynomial representation.

The diffusive terms from Eq. (5.12) are summed up in the operator

$$\mathcal{L}^{\mathrm{d},n}_{(e,\tilde{k},f)}(\check{V}^n) := D^n_{(e,\tilde{k},f)}(\check{V}^n) + E^n_{(e,\tilde{k},f)}(\check{V}^n) + F^n_{(e,\tilde{k},f)}(\check{V}^n)\,. \tag{5.25}$$

For the Navier–Stokes equations the inviscid and viscous operators are combined as

$$\mathcal{L}^{n}_{(e,\tilde{k},f)}(\check{V}^n;\check{V}^{n-1}) := \mathcal{L}^{\mathrm{e},n}_{(e,\tilde{k},f)}(\check{V}^n;\check{V}^{n-1}) + \mathcal{L}^{\mathrm{d},n}_{(e,\tilde{k},f)}(\check{V}^n) = 0\,, \tag{5.26}$$

and the complete set of these equations for all test functions is denoted $\mathcal{L}^n(\check{V}^n;\check{V}^{n-1}) = 0$.

Having derived the FE discretization of the Euler and Navier–Stokes equations, a crucial part of the numerical method is how to solve the algebraic system of equations (5.26) for the expansion coefficients $\check{V}^n$. In the generalized variable context, (damped) Newton iteration techniques have frequently been used (Hauke and Hughes, 1998; Barth, 1999) since the solution of the linear system in each iteration can benefit from the symmetrization property of the entropy variables. However, the linearization needs the flux Jacobians A^V_i, which are nontrivial to derive, implement, and evaluate. In the DG context also the linearization of the numerical fluxes would be required. Based on previous work by van der Vegt and van der Ven (2002b) and van der Ven and van der Vegt (2002), a different strategy is pursued here, namely pseudo-time integration. To apply this technique, Eq. (5.26) is considered dependent on an additional (hypothetical) time variable, τ, in the form

$$|\bar{\mathcal{K}}^n_e|\,P\,\frac{\partial\check{V}}{\partial\tau} = -\frac{1}{\Delta t}\mathcal{L}^n_{(e,\tilde{k},f)}(\check{V},\check{V}^{n-1})\,, \tag{5.27}$$

where the volume of the space element $|\bar{\mathcal{K}}_e^n|$ approximates the mass matrix,[7] and the division by the time step Δt facilitates the computation of steady state solutions and restores dimensional consistency (van der Vegt and van der Ven, 2002b). The presence of the matrix P will be explained in Section 5.2.10. The solution $\check{V}^n$ of (5.26) is obtained as the steady state of (5.27). The steady state is computed by advancing in pseudo-time τ with a Runge–Kutta (RK) method.

5.2.9 Runge–Kutta methods

In this section, a short overview is given of the Runge–Kutta methods that are used for the integration in pseudo-time. The notation is established[8] and the principle of the stability analysis carried out later on is described. For further information see (Hairer et al., 1993; Hairer and Wanner, 1993).

A discrete solution for an ordinary differential equation (ODE) problem of the type

$$y' = f(t,y), \quad t \geq 0, \quad y(0) = y_0\,, \tag{5.28}$$

at the time values t_n is obtained with an explicit RK method by constructing from the state y_n at time t_n a sequence of ν stages ξ_i, $i = 1, \ldots, \nu$, as

$$\xi_1 = y_n, \quad \xi_i = y_n + h_{\mathrm{RK}} \sum_{j=1}^{\nu} a_{ij}\, f(t_n + c_j\, h_{\mathrm{RK}}, \xi_j)\,, \tag{5.29}$$

with the real coefficients a_{ij} and c_j, $j = 1, \ldots, i-1$, and the time step length h_{RK}. The approximation for the next time level is then computed as

$$y_{n+1} = y_n + h_{\mathrm{RK}} \sum_{j=1}^{\nu} b_j\, f(t_n + c_j\, h_{\mathrm{RK}}, \xi_j)\,, \tag{5.30}$$

with real coefficients b_j, $j = 1, \ldots, \nu$. The coefficients of an explicit ν-stage RK method are often represented in tabulated form as

$$\begin{array}{c|cccc} c_1 = 0 & 0 & & & \\ c_2 & a_{2,1} & 0 & & \\ \vdots & \vdots & \ddots & \ddots & \\ c_\nu & a_{\nu,1} & \ldots & a_{\nu,\nu-1} & 0 \\ \hline & b_1 & \ldots & b_{\nu-1} & b_\nu \end{array}\;. \tag{5.31}$$

[7] When deriving the pseudo-time integration, one might actually start from $\mathrm{d}U/\mathrm{d}\tau + \mathrm{d}U/\mathrm{d}t + \partial F_{\bar{n}}/\partial x_{\bar{n}} = 0$, which would result in a mass matrix in front of the pseudo-time derivative.

[8] In particular, the indices used in this section do not follow the space-time notation used elsewhere. The summation convention is not implied and the range of indices for RK stages is given explicitly.

The restriction to explicit RK methods ensures that the matrix $A = (a_{ij})$ is strictly lower triangular. The methods used in the DG context often have only one nonzero sub-diagonal band. Such methods have the advantage that only one rather than several stages with solution data needs to be stored.

In one of the used RK methods, a modification proposed by Melson et al. (1993) is applied: For low values of the CFL number $\sigma_{\Delta t} = \Delta t\,|\boldsymbol{v}|/h$, the Melson correction makes the RK scheme point-implicit:

$$\begin{aligned} \xi_1 &= y_n\,, \\ (1 + a_{i(i-1)}\lambda)\,\xi_i &= y_n + \lambda\left[\sum_{j=1}^{i-1} a_{ij}\, f(t_n + c_j\, h_{\mathsf{RK}}, \xi_j) + a_{i(i-1)}\xi_{i-1}\right], \quad i = 2, \ldots, \nu\,, \end{aligned} \tag{5.32}$$

where λ replaces the time step length h_{RK}, as will be explained later. In this way, the properties of the RK method are adapted better to the operator $\mathcal{L}^{\mathsf{e},n}$. The effect will be demonstrated in the examination of stability properties of RK-methods.

Stability analysis of RK methods

To analyze the stability properties of a RK method, the right hand side of the ODE (5.28) is taken to be $f(t, y) = \mu y$ with $\mu \in \mathbb{C}$, $\Re(\mu) < 0$, and the initial value $y_0 = 1$. The solution of this problem is $y(t) = \exp(\mu t)$, which for the chosen values of μ converges in time: $\lim_{t\to\infty} y(t) = 0$. The analysis of the behavior of the RK method for this function yields the *linear stability domain* $\mathcal{D} := \{h_{\mathsf{RK}}\mu \in \mathbb{C} \colon \lim_{n\to\infty} y_n = 0\}$, which is the set of parameters $h_{\mathsf{RK}}\mu$ for which the asymptotic behavior of the discrete solution y_n is correct. The set $\mathcal{D}$ is determined numerically by computing the RK stages for a given initial value as

$$\xi_1 = 1\,, \quad \xi_i = \frac{y_n + \lambda\mu \sum_{j=1}^{i-1} a_{ij}\,\xi_j + a_{i(i-1)}\lambda\xi_{i-1}}{1 + a_{i(i-1)}\lambda}\,, \quad i = 2, \ldots, \nu\,, \tag{5.33}$$

and setting $r(\lambda\mu) := |\xi_\nu|$, the stability function of the RK method. All numbers $z = h_{\mathsf{RK}}\mu$ for which $|r(z)| < 1$ belong to the stability domain. Results obtained by such evaluations of the stability polynomial are shown in Figure 5.1 on page 85.

The examination of the stability of an ODE method for a linear system of m equations, $w'(t) = B(t)\,w(t) + g(t)$, with $B(t) \in \mathbb{R}^{m\times m}$, can be carried out in terms of the eigenvalues $\mu_j \in \mathbb{C}$, $j = 1, \ldots, m$, of B. Here the products $h_{\mathsf{RK}}\mu_j$ have to lie within the stability domain $\mathcal{D}$ of the RK method to ensure stability. For nonlinear operators stability results are often hard to obtain. The practical approach—namely to consider the eigenvalue locus of the locally linearized operator at the current state w_n—gives meaningful information as long as the operator is not too badly behaved, see (Hundsdorfer and Verwer, 2003, p. 47) for a discussion.

RK methods used for the pseudo-time integration

For convection-dominated flow, the five-stage fifth order EXI (explicit, for inviscid operator) Runge–Kutta method (van der Vegt and van der Ven, 2002b) is used together with the Melson correction. The computation of the stages $\check{V}^{(i)}$, $i = 1, \ldots, 5$, using the coefficients $[a_1; \ldots; a_5] = [0.0791451; 0.163551; 0.283663; 0.5; 1.0]$ proceeds as (cf. Eq. (5.27))

$$\begin{aligned} \check{V}^{(0)} &= \check{V}^{n-1}\,, \\ (1 + a_i\lambda)\check{V}^{(i)} &= \check{V}^{(0)} + a_i\lambda\left(\check{V}^{(i-1)} - \frac{1}{|\bar{\mathcal{K}}_e^n|}P^{-1}\mathcal{L}^n(\check{V}^{(i-1)}; \check{V}^{n-1})\right), \quad i = 1, \ldots, 5\,, \end{aligned} \tag{5.34}$$

where $\lambda = \Delta\tau/\Delta t = \sigma_{\Delta t}/\sigma_{\Delta\tau}$ incorporates the influence of the physical and pseudo-time CFL numbers, $\sigma_{\Delta t}$ and $\sigma_{\Delta\tau} = \Delta\tau\,|\boldsymbol{v}|/h$, respectively. In practice, a pseudo-time CFL number $\sigma_{\Delta\tau}$ is prescribed and used to locally deduce the pseudo-time step $\Delta\tau$, see Section 5.2.10. The influence of the Melson correction on the stability domain is documented in Figures 5.1a and 5.1b.

For diffusion-dominated flow, the EXV (explicit, for viscous operator) method by Klaij et al. (2006a), a four-stage RK method optimized for the large extent of the eigenvalues along the negative real axis (Kleb et al., 1999), cf. Figure 5.1c, is used. The coefficients for this RK method are $[a_1; \ldots; a_4] = [0.0178571; 0.0568106; 0.174513; 1.0]$. No Melson correction is applied in combination with the EXV method.

Finally, some tests were carried out with a nondiagonal RK method, RK44M, cf. (Dormand, 1996; Westhuis, 2001), which is given in the tabular form of (5.31) as

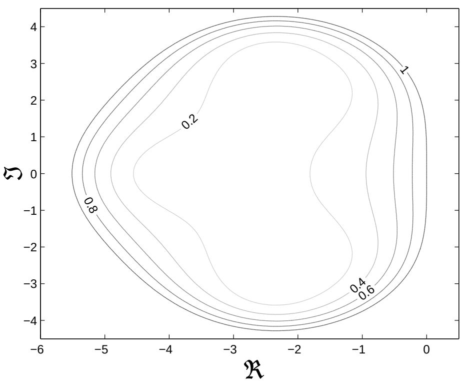
$$\begin{array}{c|cccc} 0 & 0 & & & \\ 2/5 & 2/5 & 0 & & \\ 3/5 & -3/20 & 3/4 & 0 & \\ 1 & 19/44 & -15/44 & 10/11 & 0 \\ \hline & 11/72 & 25/72 & 25/72 & 11/72 \end{array}\,, \tag{5.35}$$

in an attempt to overcome specific problems of the incompressible case, cf. Section 5.3.3. Its stability domain is shown in Figure 5.1d on the next page.

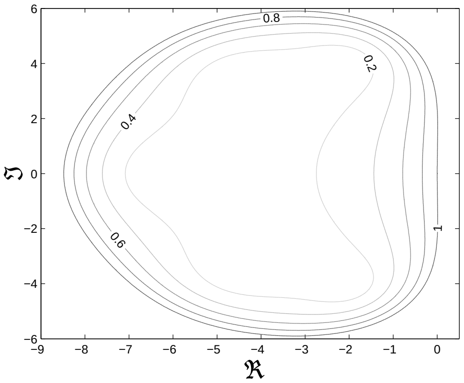
5.2.10 Application of the pseudo-time integration

Choice of the EXI RK or EXV RK method

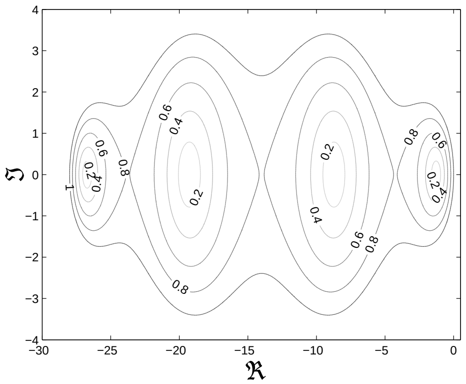
The EXI and EXV methods introduced earlier have stability domains tailored to different purposes. As the eigenvalue spectrum of $\mathcal{L}^n$ is not computed in standard simulations, an



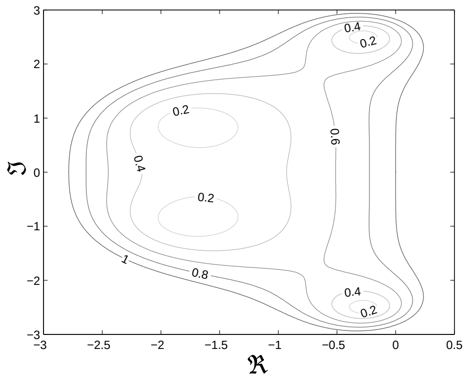
(a) EXI without Melson correction, $\lambda = 0$.



(b) EXI with Melson correction, $\lambda = 1.3$.



(c) EXV.



(d) RK44M.

Figure 5.1: Stability domains of the examined Runge–Kutta methods. Plotted are the isolines of $|\xi_\nu|$, cf. p. 83. The Melson correction applied in combination with the EXI method stretches the stability domain of the RK method further into the imaginary half plane, cf. (a) in comparison with (b). The EXV method in (c) is tailored to stiff operators with eigenvalues concentrated along the negative real axis. The RK44M method in (d) is investigated because its stability domain—unlike those of the other methods—includes a part of the imaginary axis.

ad hoc criterion has to be used to decide which of the methods to use. For the current work, the decision is based on the element Reynolds number,

$$\mathrm{Re}_{\mathrm{el}} = \frac{\rho v_{\mathrm{max}}\, h}{\eta_{\mathrm{max}}}, \tag{5.36}$$

with the maximum signal velocity v_{max}, the element diameter h, and the maximum viscous eigenvalue

$$\eta_{\mathsf{max}} = \frac{\max\{\gamma/\mathrm{Pr}, 4/3\}}{\mathrm{Re}_\eta}\,. \tag{5.37}$$

When the element Reynolds number is lower than a prescribed value, typically $\mathrm{Re}_{\mathsf{el}} = 20$, then the EXV method is applied, otherwise the EXI method. For inviscid simulations, only the EXI method is used. The same holds for incompressible cases for reasons that will be discussed later. The RK44M method is only utilized for a case study, cf. Section 5.3.3.

Determining the pseudo-time step

The goal of the pseudo-time integration is to attain a steady state in pseudo-time, which constitutes a solution of Eq. (5.26). To reach the steady state with a minimum of computational effort, the pseudo-time steps should be taken as large as possible, given the stability constraints of the RK method that is used for the integration. One of the advantages of this kind of solver is that the pseudo-time step $\Delta\tau$ is a local quantity and can vary between elements. As such, after deciding which RK method to use (see above), only the local flow properties limit $\Delta\tau$, apart from the stability properties of the RK method. These are imposed by limiting the pseudo-time CFL and von Neumann numbers,

$$\sigma_{\Delta\tau} = \frac{v_{\mathsf{max}}\,\Delta\tau}{h}\,, \qquad \delta_{\Delta\tau} = \frac{\eta_{\mathsf{max}}\,\Delta\tau}{h^2}\,, \tag{5.38}$$

respectively, by values determined by considering the eigenvalue spectrum of the Euler and viscous operators in relation to the stability domain of the RK method. For the pseudo-time CFL number $\sigma_{\Delta\tau}$ and the Euler operator $\mathcal{L}^{\mathsf{e},n}$ this analysis is the topic of Section 5.3, for other combinations see (Klaij et al., 2006a).

For the RK update, the step length is taken as the minimum of the two values arising from the limitations in (5.38),

$$\Delta\tau = \min\left\{\frac{\sigma_{\Delta\tau}\,h}{v_{\mathsf{max}}}, \frac{\delta_{\Delta\tau}\,h^2}{\eta_{\mathsf{max}}}\right\}, \tag{5.39}$$

and, finally, $\lambda = \Delta\tau/\Delta t$.

Role of the matrix P in the pseudo-time equation

It remains to explain the premultiplication of the pseudo-time derivative $\partial\check{V}/\partial\tau$ with the matrix P in Eq. (5.27). Although the augmentation of the system (5.26) with the pseudo-time derivative is an artificial step and one could argue that it should work for any variable set directly, it emerged that this is not necessarily true. In the original pseudo-time method, the conservation variables U are related to the residual $\mathcal{L}^n(U)$ in the ODE system (5.27).

If, however, the left hand side of the equations contains a different variable set, then the system can become very stiff and hardly solvable, unless extremely small pseudo-time steps are used. This effect, which occurs when the identity matrix $P = I_{(d+2)\times(d+2)}$ is used, can be undone by transforming to the conserved variable metric with $P = A_0^V$. As will be shown in Section 5.3, for regular A_0^V the resulting system has the same properties as the one for conservation variables. Unfortunately, the choice $P = A_0^V$ is not well-defined in the incompressible limit because then A_0^V is singular, independent of which variable set is used. To be able to solve the system also for the case of incompressible fluids, Chorin's idea of artificial compressibility (Chorin, 1967) is applied in the matrix A_0^V, which has the following functional form for entropy variables,

$$A_0^V = \rho^2 T \begin{pmatrix} \beta_T & \beta_T v_{\bar{\jmath}} & \beta_T(h+k) - \alpha\alpha_p T \\ & \beta_T v_{\bar{\imath}} v_{\bar{\jmath}} + \alpha\delta_{\bar{\imath}\bar{\jmath}} & [\beta_T(h+k) - \alpha(\alpha_p T - 1)]v_{\bar{\imath}} \\ \text{sym.} & & (h+k)[\beta_T(h+k) - 2\alpha\alpha_p T] + \alpha(c_\text{p} T + 2k) \end{pmatrix}, \quad (5.40)$$

with $\bar{\imath}, \bar{\jmath} = 1, \ldots, d$, see Appendix A. It is easily seen that for an incompressible medium ($\alpha_p = \beta_T = 0$) the first row and column contain only zeros. To keep A_0^V regular, its top left entry is set to a positive value ϵ in case of (near-) incompressibility. As no analysis for the magnitude of this artificial compressibility-like parameter is available, its impact is evaluated by numerical experiment, cf. Section 5.3.3. It should be emphasized that the artificial compressibility is used only for the pseudo-time integration. Unlike the direct application of the idea in the physical time derivative, the combination with the pseudo-time variable allows to obtain a time accurate solution, see also (Soh and Goodrich, 1988). In practice, the incorporation of P in (5.27) requires a premultiplication of the residual per element with P^{-1} or the solution of a $(d+2)\times(d+2)$ linear system with as many right hand sides as the number of basis functions used on the element.

For pressure primitive variables Y, the approximation of the transformation matrix A_0^Y, cf. Eq. (A.2), on page 138, takes the same route as for entropy variables. It is based on the observation that when, for the moment, only the isobaric expansion coefficient is set to zero, $\alpha_p = 0$, while retaining the isothermal compressibility β_T, the inverse of this matrix, $A_0^{Y,\alpha_p=0}$, takes the form

$$\left(A_0^{Y,\alpha_p=0}\right)^{-1} = \frac{1}{\rho}\begin{pmatrix} 1/\beta_T & (0)_{\bar{\jmath}} & 0 \\ -v_{\bar{\imath}} & \delta_{\bar{\imath}\bar{\jmath}} & (0)_{\bar{\imath}} \\ -(e^\text{tot} + p/\rho - v_n^2)/c_\text{p} & -v_{\bar{\jmath}}/c_\text{p} & 1/c_\text{p} \end{pmatrix}, \quad (5.41)$$

in which the compressibility parameter β_T occurs only once, in the top left entry. If β_T is bounded from below by a positive number ϵ, then the inverse of the resulting matrix $A_0^{Y,\epsilon}$ exists and approximates the non-existent inverse of the singular matrix $A_0^{Y,\alpha_p=\beta_T=0}$, as can be seen by multiplication with the transformation matrix for the incompressible case,

$$A_0^{Y,\alpha_p=\beta_T=0}\,(A_0^{Y,\epsilon})^{-1} = \begin{pmatrix} 0 & (0)_{\bar{J}} & 0 \\ -v_{\bar{\imath}} & \delta_{\bar{\imath}\bar{J}} & (0)_{\bar{\imath}} \\ -e^{\mathrm{tot}} + p/\rho & (0)_{\bar{J}} & 1 \end{pmatrix}, \tag{5.42}$$

$$(A_0^{Y,\epsilon})^{-1}\,A_0^{Y,\alpha_p=\beta_T=0} = \begin{pmatrix} 0 & (0)_{\bar{J}} & 0 \\ (0)_{\bar{\imath}} & \delta_{\bar{\imath}\bar{J}} & (0)_{\bar{\imath}} \\ 0 & (0)_{\bar{J}} & 1 \end{pmatrix}. \tag{5.43}$$

5.3 Stability analysis for the Euler operator

5.3.1 Description

One consequence of both the usage of a generalized variable set and the different equations of state is that the stability properties of the discretization change compared to the standard conservation variable/ideal gas case that is usually considered. Further investigation in the stability proved necessary. The approach previously applied by Klaij et al. (2006a)—considering a scalar linear advection-diffusion equation and deriving pertinent stability bounds—could not be applied in the current work. The reason is that the numerical method investigated here contains the extra transformation between the entropy or pressure primitive variables and the conservative base set, which cannot be included in a single advection-diffusion equation. Therefore, the complete discrete Euler operator $\mathcal{L}^{\mathrm{e},n}$ from Eq. (5.26) has been numerically linearized with respect to the expansion coefficients $\check{V}^n$. The base state is a two-dimensional subsonic homogeneous flow (with constant density and temperature) diagonally over the domain $\Omega = [0; 1]^2$ with periodic boundary conditions in both space directions. All computations were carried out on a mesh of 20×20 equal-sized elements.

The eigenvalues of the linearized operator $\partial\mathcal{L}^{\mathrm{e},n}/\partial\check{V}^n$ should be located in the stability domain $\mathcal{D}$ of the RK method to ensure the stability of the pseudo-time integration. The analysis of the spectrum of the matrix $\partial\mathcal{L}^{\mathrm{e},n}/\partial\check{V}$ discloses two problems: first, the use of other variable sets than the conservative variables affects the stiffness of the ODE system. This is demonstrated in Section 5.3.2 based on the ideal gas. Second, incompressibility causes the spectrum to concentrate along the imaginary axis. This issue is addressed in Section 5.3.3.

5.3.2 Compressible media

Conservative variables

The first investigation aims at the spectrum of $\mathcal{L}^{\mathrm{e},n}$ for an ideal gas, using conservative variables U at the physical CFL numbers $\sigma_{\Delta t} = 1$ and 100. In Figure 5.2 on the facing page, the eigenvalues are plotted in the complex plane together with the stability domain of the

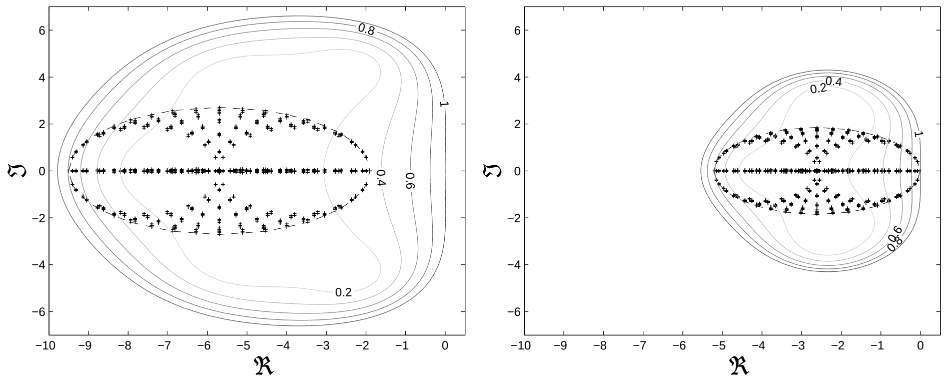
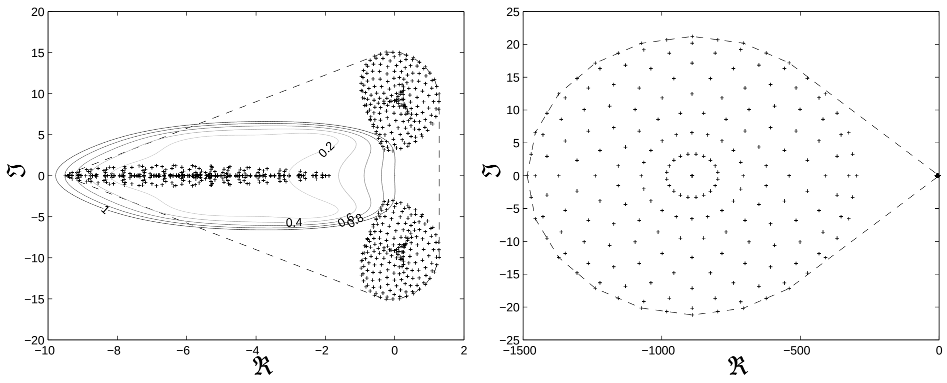


Figure 5.2: Eigenvalues ('+') of the linearized Euler operator $\mathcal{L}^{e,n}$ for two different physical CFL numbers $\sigma_{\Delta t}$. The convex hull of the spectra appears as dashed line; the stability domain of the EXI method is plotted with solid lines indicating the damping factor. The pseudo-time CFL number $\sigma_{\Delta\tau}$ has been adapted to locate all eigenvalues inside the stability domain of the EXI method.

EXI Runge–Kutta method defined in (5.34). The pseudo-time CFL number $\sigma_{\Delta\tau}$ is chosen such that the eigenvalues remain within the linear stability domain. Note the stretching of the stability domain along the negative real axis in the low CFL number case due to the Melson correction. This initial test serves to verify the results of Klaij et al. (2006a) regarding the stability of the pseudo-time integration. Note that Klaij et al. obtained their results by applying a DG discretization to the advection-diffusion equation and inserting Fourier-modes, an entirely distinct approach from the one taken here. Nevertheless the results are in the same range: where Klaij et al. gave the maximal pseudo-time CFL numbers $\sigma_{\Delta\tau} = 1.6$ and 1.8 for the physical CFL numbers $\sigma_{\Delta t} = 1$ and 100, respectively, the current method yields $\sigma_{\Delta\tau} = 1.9$ and 1.3. For practical cases, the pseudo-time step has to be limited more strictly owing to stronger nonlinearity and the presence of boundary conditions; for applications as those in Section 5.4, the pseudo-time CFL number is chosen in the range $\sigma_{\Delta\tau} = 0.8 \ldots 1.0$. From now on, the discussion focuses on the effects of introducing a different variable set. The analysis is conducted for the time-accurate case, $\sigma_{\Delta t} = 1$. The overall effects are the same for $\sigma_{\Delta t} = 100$. For the most part, the discrete expansions of all variables are limited to the constant basis function, $p_{\mathrm{t}}^{n,e} = p_{\mathrm{s}}^{n,e} = 0$, but a comparison with higher order is given, too.

Pressure primitive and entropy variables with $P = I_{(d+2)\times(d+2)}$

As one of the goals here is to use a variable set that has a well-defined incompressible limit, the discretization with pressure primitive and entropy variables is examined now. Initially,



(a) Pressure primitive variables: The spectrum extends to the positive real axis.

(b) Entropy variables: The problem becomes extremely stiff.

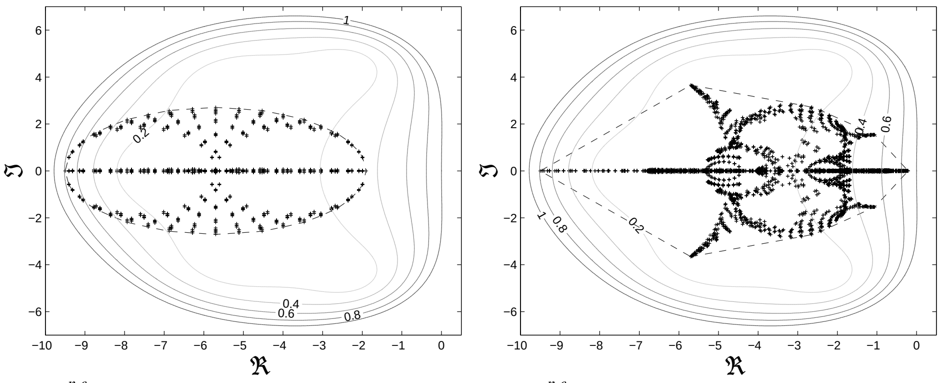
Figure 5.3: Eigenvalues for pressure primitive and entropy variables with $P = I_{(d+2)\times(d+2)}$. Note the different scales on both real and imaginary axes.

the nonlinear system is treated as if solving directly for the concerning set of unknowns, hence P in Eq. (5.27) is set to $P = I_{(d+2)\times(d+2)}$, the $(d+2)\times(d+2)$ identity matrix. The spectra of the resulting Euler operators for the same values $\sigma_{\Delta t} = 1$ and $\sigma_{\Delta\tau} = 1.9$ as before are shown in Figure 5.3. Apparently, these are of little use for computational purposes: For pressure primitive variables the spectrum extends into the (unstable) right half-plane. For entropy variables, on the other hand, the eigenvalues stay in the left half plane, but extend extremely far away from the origin, outside the stability domain of Runge–Kutta methods for any reasonable pseudo-time step. Hereby the need for restoring the eigenvalue structure of the operator in terms of conservation variables becomes evident. This is equivalent to preconditioning with the inverse of the Jacobian of the transformation from entropy to conservation variables.

Pressure primitive and entropy variables with $P = A_0^V$

By using the Jacobian $A_0^V = \partial U/\partial V$ of the transformation $U(V)$ as the 'preconditioner' matrix P, the same operator as in the conservation variable case should be recovered. This is confirmed by the spectrum plotted in Figure 5.4a on the facing page. It is computed using entropy variables with the residual transformation by $(A_0^V)^{-1}$, and it coincides with the eigenvalue structure for U-variables in Figure 5.2a on the previous page. For pressure primitive variables the introduction of $P = A_0^Y$ leads to the same result (not shown).

Incidentally, Figure 5.4 shows a result for the eigenvalues when using bilinear basis functions in space ($p_s^{n,e} = 1$). It supports that the same bound for the pseudo-time CFL number $\sigma_{\Delta\tau} = 1.9$ that originated from the analysis for constant representation per element also holds for a second order accurate computation.

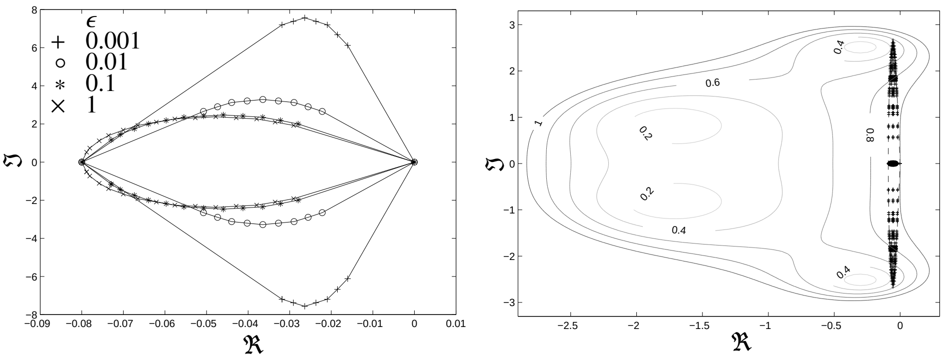


(a) $p_{\mathrm{S}}^{n,e} = 0$, compare with Figure 5.2a on page 89.
(b) $p_{\mathrm{S}}^{n,e} = 1$, i.e., four basis functions per element.

Figure 5.4: Eigenvalue spectra for the method using entropy variables V, but with restored conservation variable metric, $P = A_0^V$. Comparison for different spatial representation order.

5.3.3 Incompressible media

In Section 5.2.10, the singularity of A_0^V in the incompressible limit has been discussed, along with a strategy how to avoid the resulting problems by introducing an artificial compressibility-like parameter ϵ in the matrix. Generally, the eigenvalues of the incompressible operator lie very close to the imaginary axis so that the pseudo-time step $\Delta\tau$ is limited primarily by the imaginary extent of the spectrum. When this is the case, the introduction of the artificial compressibility parameter ϵ in A_0^V can help alleviate the stability restriction. In Figure 5.5a on the following page, the convex hulls of the eigenvalue sets computed with different values of ϵ are plotted. The pseudo-time CFL number $\sigma_{\Delta\tau} = 0.016$ is the same for all shown sets. The figure substantiates that the extent in the imaginary direction is decreased by approximately a factor of 3 when setting $\epsilon = 1$ instead of $\epsilon = 0.001$. At $\epsilon = 1$ the effect saturates and no further advantage is gained. For smaller values of ϵ the eigenvalues spread out further along the imaginary axis. The introduction of ϵ does not degrade the (time) accuracy, because it affects only the pseudo-time process and the equations solved in physical time are still the original Euler equations. Figure 5.5b shows the alternative Runge–Kutta method, RK44M introduced in Section 5.2.9, which may help to overcome some of the problems caused by the proximity of the eigenvalues to the imaginary axis. The stability domain of the EXI method only touches the imaginary axis in the origin and then turns away from it, which leads to the predominant restriction in the incompressible case. In contrast, the stability domain of RK44M runs along the imaginary axis and even contains a small part of the right half plain. Thereby the eigenvalues are contained in the stability domain even if they are very close to the imaginary axis. The problem that the time step is limited by the extent along the imaginary axis, however,



(a) Convex hulls of the eigenvalue spectra for different values of ϵ.

(b) $\sigma_{\Delta t} = 1$, $\sigma_{\Delta\tau} = 0.018$; stability domain of the RK44M method.

Figure 5.5: Examination of the eigenvalue spectrum of the Euler operator $\mathcal{L}^{e,n}$ for entropy variables with $P = A_0^V$ and incompressible flow.

remains. Runge–Kutta methods that stretch out further in this direction typically have more stages and are more expensive to compute (Hairer et al., 1993; Hairer and Wanner, 1993), so that no runtime advantage can be expected. Improvements hence should concern the structure of the spectrum, but further research is necessary on this topic. The inviscid test cases in Section 5.4 were all computed using the EXI method.

5.4 Numerical examples

The discontinuous Galerkin method described in this chapter has been implemented based on the object-oriented framework hpGEM (cf. Chapter 6). To complete the inspection of the numerical method and to verify the implementation, results of computations for several flow problems are added. The main emphasis is to demonstrate applicability for different media, e.g. compressible and incompressible fluids, when using entropy or primitive variables. The discretization for the stationary test cases uses linear in space and constant in time basis functions, unless stated otherwise. For the (supersonic) gas flows preference is given to the HLLC flux. In the compressible/incompressible comparison, for reasons discussed in Section 5.2.5, HLLC is replaced by the LLF flux. The ideal gas tests assume the adiabatic index to be $\gamma = 1.4$. All computations are based on entropy variables unless explicitly mentioned otherwise.

First, a series of inviscid test cases is presented, starting with an example concerning low Mach-number flow through a channel with a bump, see Section 5.4.1. Supersonic flow is the topic of the following two test cases, of which one uses an ideal gas, cf. Section 5.4.2 and the other one a non-ideal gas, see Section 5.4.3. In Section 5.4.4, results are provided for compressible and incompressible inviscid flow around a circular cylinder. Next, the exact solution of the incompressible Navier–Stokes equations by Kovasznay is used to assess the discretization of the viscous terms in Section 5.4.5. Viscous flow at different Reynolds numbers is also considered in Section 5.4.6, where the efficiency of the iterative solver is evaluated. Finally, in Section 5.4.7, some remarks are added about the relative computational cost of using the different variable sets.

5.4.1 Ideal gas: channel with a bump

Subject of this test case is subsonic flow in a channel of unit width with a 10% circular bump in the central third of the simulated length (Bijl and Wesseling, 1998). The grid is boundary fitted with 63×22 cells. Results are given for the inflow Mach numbers $M_\infty = 1.0 \cdot 10^{-6}$, and $M_\infty = 0.5$. At the inflow, entropy, stagnation enthalpy and the flow angle are prescribed, while at the outflow only the pressure is specified (Darmofal et al., 2000). Along the upper and lower channel wall, a slip flow boundary condition is used.

For subsonic flow, the Mach number fields should be symmetric with respect to the middle of the channel ($x_1 = 1.5$). This is confirmed by the results in Figure 5.6 on the next page, with a small deviation for the higher inflow Mach number in the region downstream of the bump. The plots show the same contour levels as used by Bijl and Wesseling (1998) for this test case, and very good agreement with their results is observed.

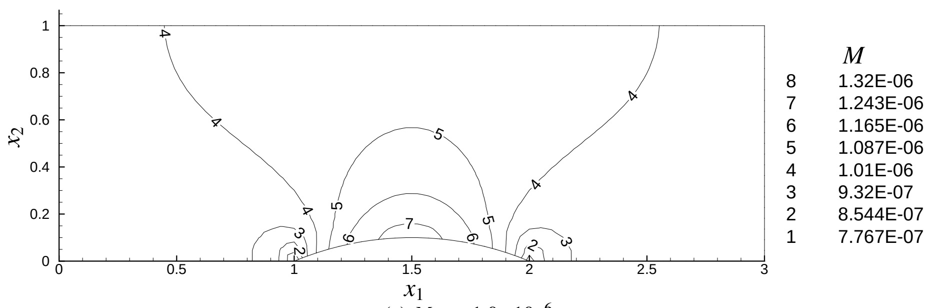
5.4.2 Ideal gas: oblique shock

To give evidence of the capabilities of the developed method, a result for the supersonic inviscid test from Section 4.5.3 is added. For the details of the setup see page 62.

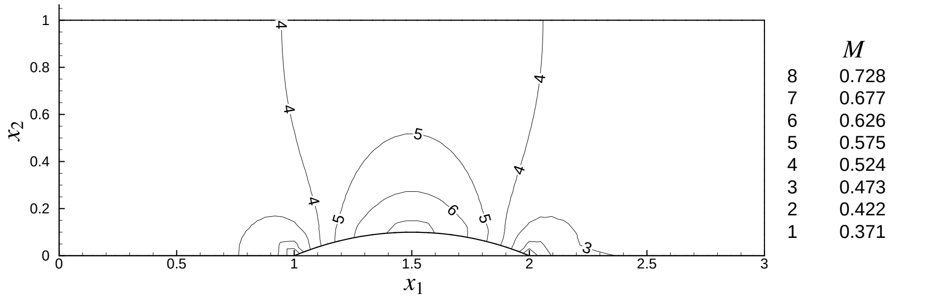
The pressure field including the shock is plotted in Figure 5.7a on page 95. Note that no discontinuity capturing or stabilization is used here, hence the overshoot close to the shock. In this way, it is also reasonable to compare the results for different sets of variables, cf. Figure 5.7b. Only minor differences are observed, after all the equations that are solved are always the same set of conservation equations (2.14). The small differences that are visible arise because of the differing discrete solutions when expanding in terms of the various variable sets and due to the transformations between the conservative or entropy variables and the plotted variable p. Independent of the variable set the location of the shock is correctly represented, cf. (Hauke and Hughes, 1998).

5.4.3 Real gas: shock-tube problem

A particular instance of a real gas is the covolume gas, a vdW gas with $a = 0$, cf. Section 2.7.2. Toro (1989) posed a shock-tube problem for such a medium with the following initial data: left state $\rho^\mathsf{L} = 1$, $v^\mathsf{L} = 0$, $p^\mathsf{L} = 1$, right state $\rho^\mathsf{R} = 0.125$, $v^\mathsf{R} = 0$, $p^\mathsf{R} = 0.1$, separated at $x = 0.5$, for a gas with the (unrealistically high) covolume $b = 0.8$,



(a) $M_\infty = 1.0 \cdot 10^{-6}$



(b) $M_\infty = 0.5$

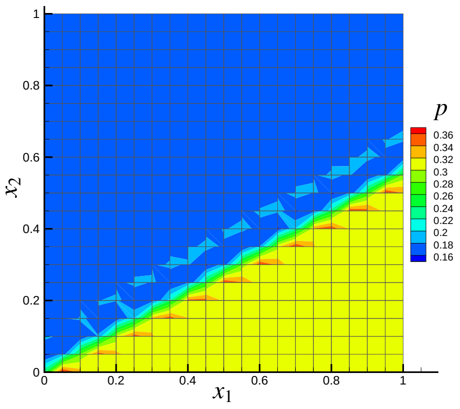
Figure 5.6: Mach number fields for two different inflow Mach numbers M_∞ into a channel with a 10% circular bump, computed on a grid with 63×22 elements and $p_{\mathrm{s}}^{n,e} = 1$.

cf. Eq. (2.34b) on page 23. The adiabatic index is $\gamma = 1.4$. It is well-known that the performance of the LLF flux is inferior to Riemann-problem based methods on such test cases. The HLLC flux is applicable to covolume gases and allows to obtain a result of comparable quality as with more specialized methods.

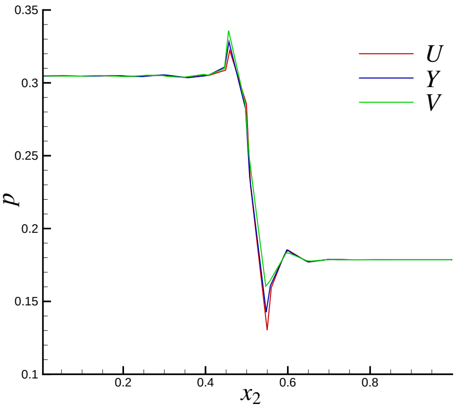
This test highlights the wide applicability of the generalized variable formulation of the Euler equations (here using V-variables) and the implementation. The solution of the shock-tube problem for a covolume gas differs substantially from the ideal gas case, cf. Figure 5.8 on page 96, which is reflected well by the numerical results.

5.4.4 Compressible and incompressible inviscid flow around a circular cylinder

This test case considers the flow of an ideal gas (subsonic) and an incompressible fluid around a fixed circular cylinder with unit radius. The far-field flow is aligned with the x_1-axis, constant, and homogeneous: $\boldsymbol{v}_\infty = u_\infty \boldsymbol{e}_1$. In the limit of vanishing Mach number, $M_\infty \to 0$, the flow field is given by the classical potential flow solution. Potential flow denotes the idealization of a flow as inviscid, incompressible, and irrotational. Under these



(a) Pressure field computed using entropy variables.



(b) Comparison of the computed pressure fields along $x_1 = 0.9$ using different variable sets.

Figure 5.7: Numerical results for the oblique shock test case, computed on a 20×20 mesh with $p_s^{n,e} = 1$.

assumptions, the governing equations can be simplified considerably, which—for some geometries—allows to find analytical solutions. When considering the Euler equations the assumption of vanishing viscosity has already been made. An incompressible fluid with zero vorticity at an instant will never generate vorticity but remain irrotational, $\nabla \times \boldsymbol{v} = 0 \ \forall\, x, t$, and for low-speed compressible flows, density changes are small. Given these conditions, it is reasonable to use an exact solution for a potential flow case as reference for a method to numerically solve the Euler equations.

To conform to a previous study (van der Vegt and van der Ven, 2002a), the free-stream Mach number of the ideal gas flow is chosen as $M_\infty = 0.38$. At the outer boundary, the far-field state $p_\infty = 4.945$, $\boldsymbol{v} = \boldsymbol{v}_\infty$, $T = 1$, is prescribed as the external state of the numerical flux. Due to the absence of viscosity in the Euler equations, a slip flow condition is applied on the cylinder surface: $\boldsymbol{v} \cdot \boldsymbol{n} = 0$. For compressible flow, the error of the numerical method is indicated by the dimensionless total pressure loss π, defined as

$$\tilde{\pi} := 1 - \frac{p}{p_\infty} \left(\frac{1 + \frac{1}{2}(\gamma - 1)M^2}{1 + \frac{1}{2}(\gamma - 1)M_\infty^2} \right)^{\frac{\gamma}{\gamma-1}} . \tag{5.44}$$

The total pressure loss should be zero everywhere (van der Vegt and van der Ven, 2002a), deviations occur only due to the discrete approximation, which causes entropy production at the cylinder surface. Cylinder flow is also a test case for numerical methods for incompressible flow. The results are very similar to low Mach number compressible computations.

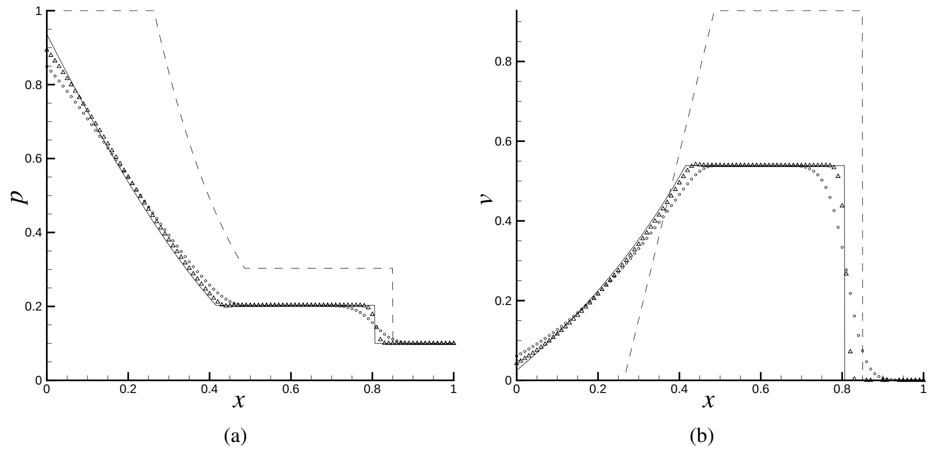


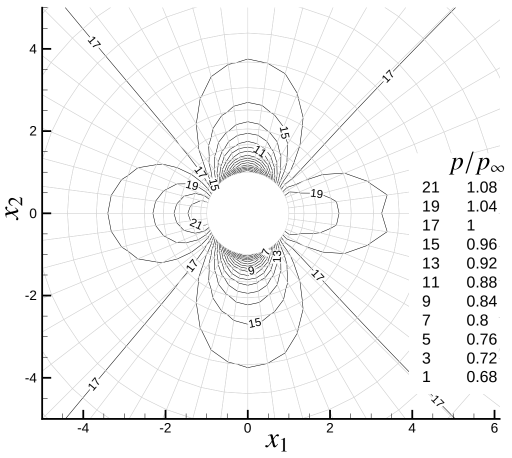
Figure 5.8: Shock-tube problem with covolume gas at $t = 0.2$, computed using a mesh of 100 elements. Symbols: $\circ$: $p_{\mathrm{s}}^{n,e} = 0$, $\triangle$: $p_{\mathrm{s}}^{n,e} = 1$, solid line: exact solution for covolume gas, dashed line: exact solution without covolume, i.e., for an ideal gas; (a) pressure p, (b) velocity v.

The pressure field (normalized by the far-field pressure p_∞) and Mach contours for $M_\infty = 0.38$ according to a computation with entropy variables using four basis functions per element on a mesh with 48×36 elements is shown in Figure 5.9 on the facing page. Note that the plots show continuous fields which are obtained by averaging the solution from neighboring elements in each node. The result should be contrasted with those presented by Bassi and Rebay (1997b): without superparametric elements, higher order discretization, and on a mesh with lower resolution than their 64×16 grid, it is possible to compute a numerical solution without the substantial unsteady wake behind the cylinder.

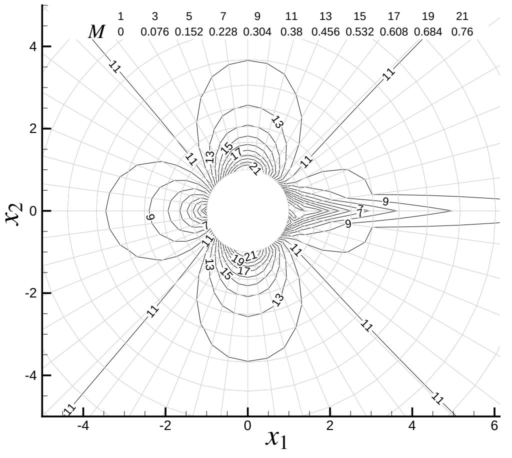
Figure 5.9c on the next page shows the total pressure loss along the cylinder surface for entropy variables using two different orders of the spatial representation (since this is a steady state case $p_{\mathrm{t}}^{n,e} = 0$ on all elements). The result using bilinear polynomials per element compares very well with the one obtained by van der Vegt and van der Ven (2002a) on a linear isoparametric mesh of comparable resolution. Results for different variable sets are hardly distinguishable in this test, so that the comparison is omitted. Repeating the same test with an incompressible fluid yields almost symmetric fields about $x_1 = 0$, cf. Figure 5.10 on page 98.

5.4.5 Incompressible viscous test case: Kovasznay flow

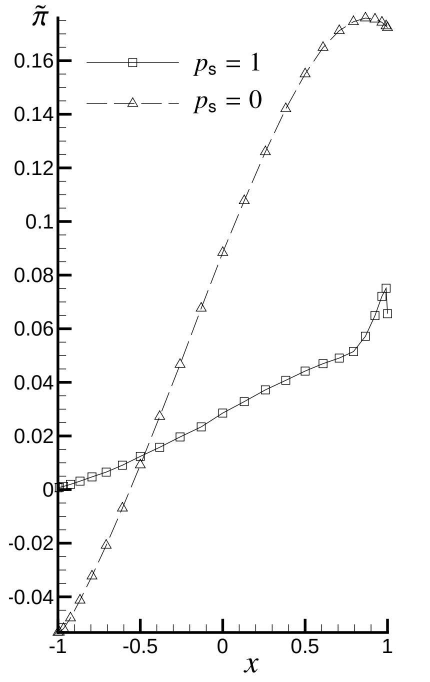
To test the discretization of the viscous flow equations, one of the few exact solutions of the two-dimensional incompressible Navier–Stokes equations is used here: Kovasznay flow. It



(a) Normalized pressure field.



(b) Local Mach number M.



(c) Total pressure loss along the cylinder surface for two orders of polynomial expansion.

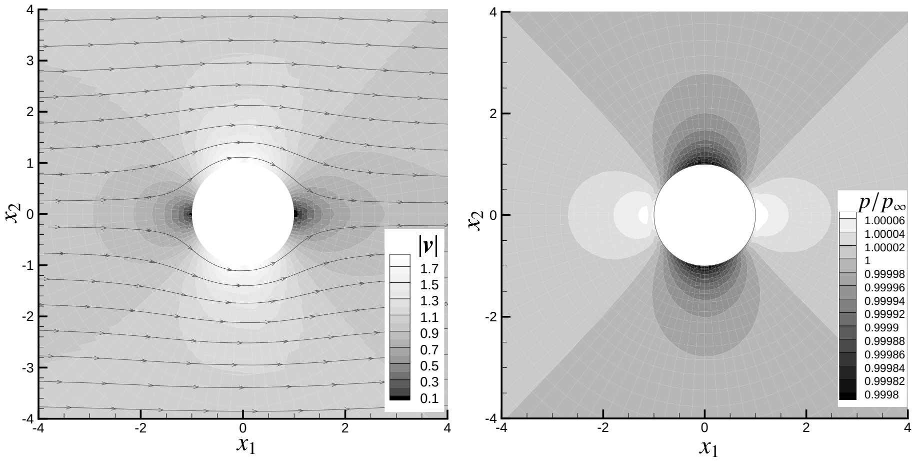
Figure 5.9: Numerical results for the flow of an ideal gas at $M = 0.38$ around a circular cylinder of unit radius. The computation uses entropy variables on a 48×36 mesh that extends to the radius $r = 56.5$.

is frequently used to quantify the accuracy of numerical algorithms, see, e.g., (Cockburn et al., 2002; Bassi et al., 2006; Mozolevski et al., 2007). The flow field is given by

$$p_{\mathsf{Kov}}(x_1, x_2) = -\frac{1}{2}\exp(2\Lambda x_1) + C\,, \tag{5.45a}$$

$$u_{\mathsf{Kov}}(x_1, x_2) = 1 - \exp(\Lambda x_1)\cos(2\pi x_2)\,, \tag{5.45b}$$

$$v_{\mathsf{Kov}}(x_1, x_2) = \frac{\Lambda}{2\pi}\exp(\Lambda x_1)\sin(2\pi x_2)\,, \tag{5.45c}$$



(a) Absolute value of the velocity and streamlines for incompressible flow around a circular cylinder.

(b) Normalized pressure field for $M = 0.01$ computed on a 96×72 grid.

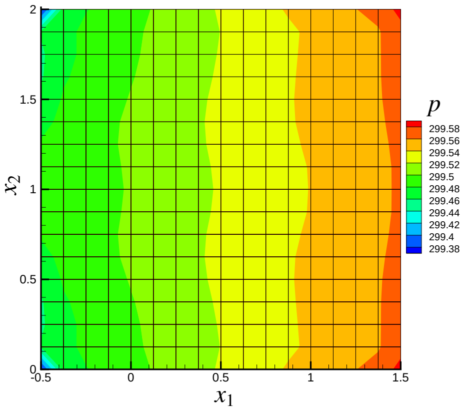
Figure 5.10: Incompressible and low Mach number flow around a circular cylinder.

with the arbitrary constant C and the parameter Λ given by

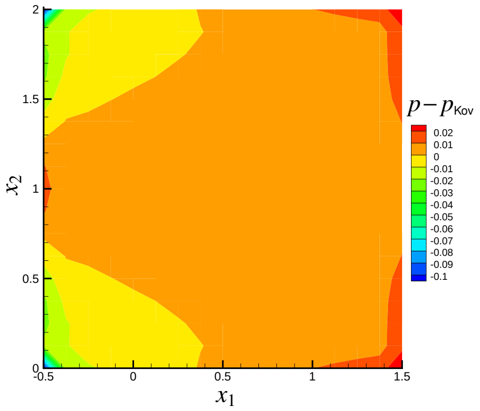
$$\Lambda = \frac{\mathrm{Re}_\eta}{2} - \sqrt{\frac{\mathrm{Re}_\eta^2}{4} + 4\pi^2}\,. \tag{5.46}$$

The flow domain is taken as $\Omega = [-1/2; 3/2] \times [0, 2]$ and the relevant components of the exact solution are prescribed according to the incompressible in- and outflow conditions on the boundaries. For this stationary solution, constant in time basis functions are used and the spatial order is set to $p_{\mathrm{s}}^{n,e} = 1$. To confirm the convergence order of the numerical method, the computation is repeated for a series of quadrilateral meshes with 4, 8, 16, and 32 elements per spatial coordinate direction. Primitive variables are used for this case and the energy equation is disregarded. The Reynolds number is chosen as $\mathrm{Re}_\eta = 200$ or $\mathrm{Re}_\eta = 1000$. A fixed number of 10^4 pseudo-time steps is taken.

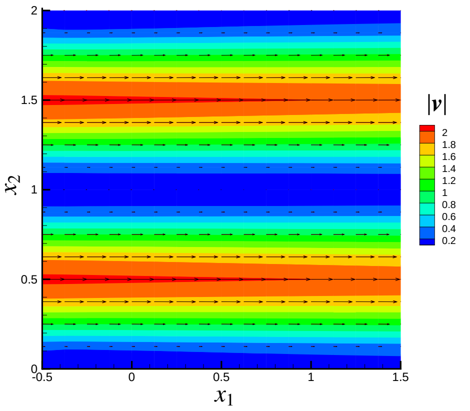
The numerical solution for $\mathrm{Re}_\eta = 1000$ obtained on the 16×16 mesh is shown in Figure 5.11 together with spatial plots of the error. The error in the computed solution seems to be mainly introduced by the boundary conditions, in particular in the pressure field at the corner points of the domain. This coincides with the observed slow convergence of the pressure field. A convergence study based on the $L^2(\Omega)$-error of the velocity field (scaled by the domain size) is contained in Table 5.1 on page 100. It confirms the second order accuracy when using bilinear basis functions.



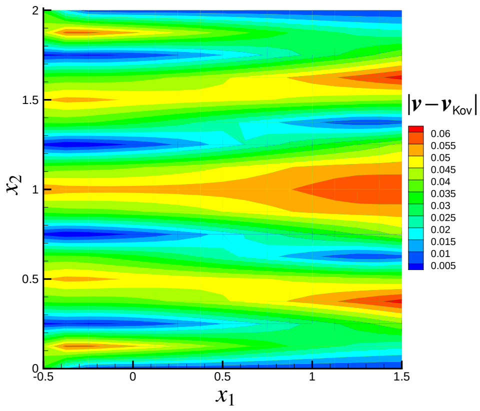
(a) Mesh and computed pressure.



(b) Error of the computed pressure.

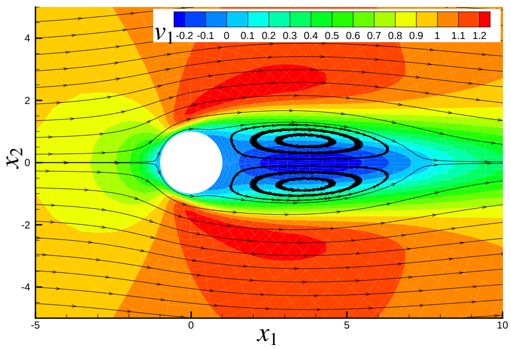


(c) Absolute value and vector field of the computed velocity.

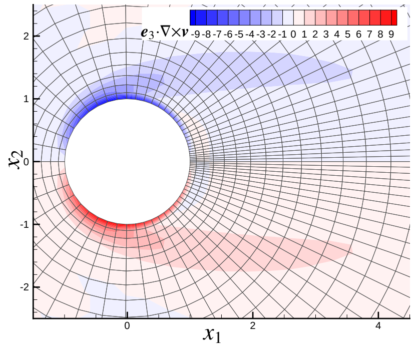


(d) Absolute value of the error of the computed velocity.

Figure 5.11: Numerical solution for Kovasznay flow (left) and the associated errors (right). The results concern the 16 × 16 mesh.



(a) Velocity component v_1 and streamlines.



(b) Vorticity contours and mesh.

Figure 5.12: Viscous ideal gas flow around a circular cylinder at $\text{Re}_\eta = 40$.

	$\mathrm{Re}_\eta = 200$		$\mathrm{Re}_\eta = 1000$	
h	$\lVert \boldsymbol{v} - \boldsymbol{v}_{\mathrm{Kov}} \rVert_{L^2(\Omega)}$	order	$\lVert \boldsymbol{v} - \boldsymbol{v}_{\mathrm{Kov}} \rVert_{L^2(\Omega)}$	order
$5.00 \cdot 10^{-1}$	$1.492 \cdot 10^{-1}$		$5.808 \cdot 10^{-1}$	
$2.50 \cdot 10^{-1}$	$9.945 \cdot 10^{-2}$	0.59	$1.023 \cdot 10^{-1}$	2.51
$1.25 \cdot 10^{-1}$	$3.848 \cdot 10^{-2}$	1.37	$2.417 \cdot 10^{-2}$	2.08
$6.25 \cdot 10^{-2}$	$9.946 \cdot 10^{-3}$	1.95	$5.846 \cdot 10^{-3}$	2.05

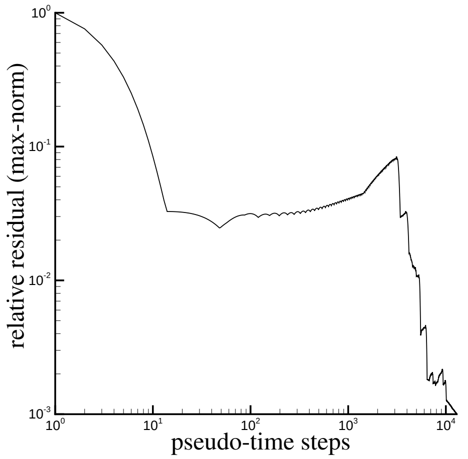
Table 5.1: Error of the computed velocity field for Kovasznay flow with the Reynolds numbers $\mathrm{Re}_\eta = 200$ and $\mathrm{Re}_\eta = 1000$.

5.4.6 Compressible viscous flow around a circular cylinder

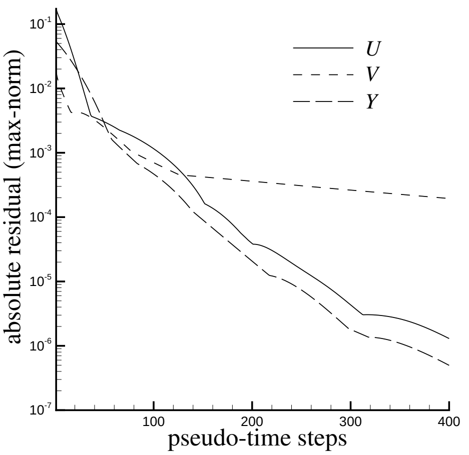
The flow around a circular cylinder considered in Section 5.4.4 also leads to interesting results when viscous effects are included. In this case, the resulting flow pattern also depends on the Reynolds number. At low Reynolds numbers, the main flow remains steady, separating from the cylinder. In the region downstream of the cylinder a pair of counter-rotating stationary vortices forms. For higher Reynolds numbers, the flow becomes unsteady and periodic vortex shedding occurs.

The subsonic flow of an ideal gas at the free-stream Mach number $M = 0.3$ is considered. The Reynolds number is set to $\mathrm{Re}_\eta = 40$ and the other dimensionless parameters are chosen as in the inviscid test. The v_1-velocity, streamlines and vorticity are plotted in Figure 5.12 on the preceding page. This result has been obtained by computing a single time step of length $\Delta t = 10^9$ on a mesh of 64×64 elements with $p_\mathrm{s} = 1$ and Y-variables. The flow separation and the region with recirculating flow are clearly visible. The convergence of the system residual (relative to the initial residual) during the pseudo-time integration of the steady-state case is shown in Figure 5.13a on the next page. In the first twenty solver iterations the residual decreases by approximately two orders of magnitude. Only after a long period of stagnation can the pseudo-time iteration reduce the residual further, presumably when the waves excited by the initial and boundary conditions have left the domain. The slow convergence for high CFL numbers $\sigma_{\Delta t}$ can be attributed to the eigenvalues of the operator being concentrated close to the imaginary axis, see Figure 5.2 on page 89 for the basic effect. Here, $\sigma_{\Delta t}$ is much larger, which explains the slow convergence.

By contrast, in the time-dependent case the pseudo-time integration is much more successful at reducing the residual. Figure 5.13b provides evidence for this claim: for a time step of $\Delta t = 0.25$, the residual is reduced faster and to a much larger extent than in the steady-state case. The data stems from a simulation with the Reynolds number $\mathrm{Re}_\eta = 200$ on a mesh with 80 elements in the angular and 84 in the radial direction. For these settings the figure compares the residual for all three variable sets, U, V, and Y. Conservation and pressure primitive variables yield very similar results in this case. For entropy variables, the initial residual is slightly smaller, but in the course of the pseudo-time integration its reduction lags behind the reduction of the residual of other variable sets. In practice for time-dependent cases a residual reduction of two orders of magnitude is required in each time step. This goal is typically reached after about 100 iterations, so that the difference in the long-term behavior—though strikingly large in this case—does not play a role.



(a) Steady state problem for $\mathrm{Re}_\eta = 40$. The diagram shows the residual convergence during the iterations of the nonlinear solver using pressure primitive variables for a single time step with $\Delta t = 10^9$.



(b) Unsteady problem for $\mathrm{Re}_\eta = 200$. Comparison of the typical residual convergence in pseudo-time for a single physical time step with $\Delta t = 0.25$ using different variable sets.

Figure 5.13: Test case viscous ideal gas flow around a circular cylinder: residual convergence during the pseudo-time integration.

5.4.7 Runtime comparison

Finally, an indication of the relative cost of using the different variable sets is given on the basis of computations with the Euler equations. It is difficult to compare on the level of a complete simulation as the magnitude and convergence of the residual $\mathcal{L}^{\mathrm{e},n}(V)$, which is used as stopping criterion for the pseudo-time integration, differs between variable sets, and it is not clear how to compare residual levels. Errors may be amplified by the discontinuous discretization, see Figure 5.7b, so that again an exact comparison is hard to realize. What one can compare easily, however, is the cost per pseudo-time step, unrelated to its effect on the convergence. For the Euler operator, when defining the cost of a pseudo-time step using U-variables as 1.0, the relative cost of one step with pressure primitive variables Y is approximately $1.0 \ldots 1.1$ and for entropy variables V it is in the range $1.6 \ldots 1.7$. These are indications that depend on several details of the method, such as the numerical flux, the Runge–Kutta method, the order of the expansion per element, and the thermodynamical relations of the medium considered. Two contributions account for the extra cost: First, the transformation that has to take place each time a specific set of quantities is used, e.g. conservation variables and pressure in the flux evaluation. Second, the residual transformation with $P = A_0^V$ introduces additional effort.

The cost of pressure primitive variable computations is surprisingly low, a pseudo-time step takes approximately the same time as one with conservative variables. Presumably, this effect is rooted in the extra computations that are necessary also when conservative variables are used (e.g., to obtain the pressure for the flux evaluation). The transformations for pressure primitive variables are also comparably simple, mostly rational functions. For entropy variables, by contrast, the variable transformations typically use more complex functions (this depends on the thermodynamics of the medium of course). Furthermore, in the software implementation, the residual transformation is realized as a linear system solution for entropy variables, but as a direct multiplication of the residuals with $(A_0^Y)^{-1}$ for pressure primitive variables. This difference is rooted in the simpler structure of A_0^Y and presumably accounts for a considerable part of the difference.

Ultimately, the consideration of the computational cost has to be combined with an appraisal of the benefits of different variable sets, cf. Section 2.8. Entropy variables constitute an interesting choice because of their theoretical properties while a consideration solely based on computational efficiency suggests preferring pressure primitive variables.

5.5 Conclusions

A discontinuous Galerkin finite element discretization of the compressible Euler equations (van der Vegt and van der Ven, 2002b) has been extended in two respects: First, the viscous terms of the Navier–Stokes equations have been added using the interior penalty approach. This addition allows to simulate processes involving compressible Newtonian fluids. Second, by allowing to solve for generalized variable sets, the method has been made more flexible regarding the accommodated thermodynamical fluid models. For this step, some parts of the solution algorithm of the original method, which is only applicable to (ideal) gases, had to be adapted. In particular the working of the pseudo-time stepping method used to solve the nonlinear algebraic system of equations has been investigated; with a residual transformation and exploiting an artificial compressibility-like idea, the method is suitable for entropy and pressure primitive variables. For incompressible media, the efficiency of the pseudo-time integration admittedly falls behind the expectations raised by compressible applications. Hence, further work should aim at improving the efficiency of the nonlinear solution step using a multigrid algorithm (van der Vegt and van der Ven, 2002b; Klaij et al., 2007). Alternatively, one might try to adapt the Newton method from Chapter 4, though it will be more difficult to apply in more complex situations as the envisaged flows with several fluids and free interfaces.

In principle, however, the simulation of various physical flow conditions is possible with the presented algorithm. Rather than being limited to the typical gas dynamics regime, for entropy and pressure primitive variables the low Mach number limit of the Navier–Stokes equations is well-defined and even computing the flow of incompressible fluids is possible as has been demonstrated by several numerical test cases originating from different fields in compressible and incompressible fluid dynamics. Finally, an estimate of the difference in the numerical cost incurred by the use of different variable sets has been given. Altogether, the findings allow a favorable cost-benefit evaluation for the usage of the generalized variable formulation.

Entropy variables have been given specific consideration, because of their theoretically appealing properties. In this respect, the numerical method developed in this chapter is on par with the Galerkin least-squares FEM from Chapter 4. The advantage of the discontinuous Galerkin method lies in large part in its potential regarding (moving) unstructured meshes. Also the straightforward implementation of many boundary conditions benefits its application. At this stage, a positive result regarding the first two central questions formulated in Chapter 1 has been achieved: The applicability for different media and physical conditions has been combined with a discretization technique that makes available the utmost of geometric flexibility. In combination with an interface tracking algorithm, the numerical method discussed in this chapter is suitable to compute multiphase flow taking into account the different thermodynamical properties of liquids and gases.

Chapter 6

hpGEM – A software framework for discontinuous Galerkin finite element methods

Wij zouden anders steeds van voren af aan
moeten beginnen en de continuïteit,
die een van de wezenskenmerken
van het leven der wetenschap is,
zou daarmee verloren gaan.

Jan Marius Romein (1893–1962),
In Opdracht van de Tijd

6.1 Introduction

The development of a finite element method for a given set of partial differential equations is a complex task and typically implies the conversion of the mathematical method into a computer program. It is notable that the mathematical formalization of FEMs is largely independent of the specific application and that the same is valid for the resulting software. This realization is at the heart of hpGEM, a framework for the implementation of finite element methods, which is described in this chapter.

Focusing on discontinuous Galerkin methods, data structures and algorithms are presented that are common to many FEMs and their implementation as components of an object-oriented framework is discussed. This framework facilitates and accelerates the implementation of finite element programs, the assessment of algorithms, and their application to real-world problems.

Given hpGEM, it remains to the applying scientist—starting from a correct mathematical formulation of the discrete problem—to (i) assemble the provided components in a correct and efficient way, and (ii) add whatever is special or unique to a considered problem or algorithm. Obviously additions to the framework are possible when sufficient generality and usefulness have been shown. However, it is not the goal of hpGEM to be a ‘solver’ for a predefined type or set of equations. Rather it is intended to serve numerical scientists by providing the geometric and functional concepts that occur in a worked-out weak form. Concrete problems are solved by sample applications, but the actual hpGEM framework concentrates on the general building blocks.

This chapter contains and extends the material presented in (Pesch et al., 2007). The ideas and requirements on which hpGEM is built are explained in Section 6.2. Software design, implementation, and handling issues are touched upon in Section 6.3. The design of a number of components of the framework is exemplified in Section 6.4 and case studies from the implementation of the Navier–Stokes solver of Chapter 5 are added in Section 6.5. Section 6.6 concludes the chapter and provides an outlook on the challenges for the further development of hpGEM.

6.2 General ideas

Based on the considerations given in Section 1.4 of the introductory chapter and on the presentation of two finite element methods in Chapters 4 and 5, the following basic requirements for the software framework are formulated as follows:

- Ability to use discontinuous Galerkin methods.
  These methods are a focus of current research. From Chapter 5 it follows that some aspects make DG methods different from continuous finite element methods. In particular the use of face-based data structures (e.g., for the computation of numerical fluxes in systems of hyperbolic conservation laws) is different from classical FEMs. On the other hand, some of these differences can be taken advantage of, in particular in the form of increased concurrency in the computation of the element-based discretization.

- General types of elements in $\mathbb{R}^d, d = 1, \dots, 4$.
  Another advantage of DG methods is their flexibility with respect to the geometric shape of the elements. For two-dimensional meshes, mixtures of triangles and quadrilaterals can be as easily accommodated as three-dimensional meshes with tetrahedra, pyramids, triangular prisms, and hexahedra. Applications of discontinuous Galerkin methods on mixed meshes are presented in (Pesch et al., 2007). For so-called space-time FEMs, one has to extend the elements with an extra dimension for time, so for three-dimensional space the elements become four-dimensional.

- Dimension-independence of the code.
  For many FE algorithms the mathematical formulation is independent of the dimension of the problem. To a large degree this property can be preserved by the software environment. Application codes developed in a two-dimensional setting can frequently be used for three-dimensional computations by changing a (C++-template-) parameter, if the user's code allows this, too.

- Easy and fast generation of applications.
  When testing an algorithm the first question is how long it will take to correctly

implement it. Hence it is important that provided components are easy to use and well-documented.

- Parallelism handled internally.
  While parallel computing is of prime importance for many relevant physical applications, it is a corollary of the previous item that the exploitation of parallel computation strategies should be as far as possible handled internally, so that a user program can remain largely unchanged for serial and parallel processing. The realization of this requirement is still future work.

- Enforcement of quality standards.
  To guarantee a correct and extensible FE framework, documentation is added both in-code and externally. What has proven even more valuable is the test-suite built up with the code. Unit tests check the correct working of individual classes, procedures, and collaborative tasks (like computing normal vectors, evaluating integrals). The unit tests can be re-run at any time, so that changes and additions to the framework can immediately be assessed for correctness.

- Access to external software for common tasks.
  Because the focus of hpGEM is on FE-related research, existing components for the pre- and post-processing steps are used where possible, e.g., for mesh generation and visualization. Also for linear algebra tasks, existing high-quality solutions can be employed by the framework or made accessible to the user.

The above items constitute the most important requirements, yet the result of a search for a matching candidate was that no available solution was satisfying. Clearly, failure to comply with the first two items from the requirements list leads to the rejection of many available FE packages. The other topics are more debatable, but collectively can be of equal importance. This conclusion is briefly discussed by comparing a few available software packages based on object-oriented development to the above list; the findings are typical for many other FEM software artifacts, which are not reviewed here.

Discontinuous Galerkin methods, being a relatively recent topic, are not implemented by most FE packages and their special aspects (e.g., face data structures) are not taken into account. Packages which support discontinuous Galerkin methods are `deal.II` (Bangerth et al., 2006) and DOLFIN (Hoffman and Logg, 2002). However, the former works only on n-cubes, i.e., elements are lines, quadrilaterals, and hexahedra in dimension 1, 2, and 3, respectively. The latter, just like other packages, e.g. ALBERTA (Schmidt and Siebert, 2006), restricts itself to simplices, so that meshes consist of triangular or tetrahedral elements. Mixed meshes are rarely accommodated by existing FE software, possibly because of the complications when using continuous FEMs. No package could be found that offers support for space-time discretizations, elements are implemented only for dimensions one to three. Within these bounds, codes based on `deal.II` or ALBERTA can be dimension-independent for meshes based exclusively on *n*-cubes or simplices, respectively.

Different philosophies become apparent when it comes to the application program interface (API) available to the users of a software framework. In some packages, e.g. DOLFIN, high-level access is provided to a degree that modules allow the solution of pre-programmed equations by choosing the geometry, suitable initial and boundary conditions.

The access to external software in the compared packages is—if at all provided—frequently hidden by extensive interfaces. This has the advantage of unified access to different packages, which, however, comes at high development cost and possibly restricts the range of usable options of the integrated packages. Remarkably, some FE packages rely entirely on their own developments regarding linear algebra and equation solvers.

Efforts regarding the quality measures mentioned previously vary; online documentation for the abovementioned packages is available, sometimes also detailed tutorials, e.g. for `deal.II`, which also seems to be most advanced regarding its test suite.

6.3 Choice of programming paradigm and language

When dealing with FEMs, an advantageous aspect of their structure is the way in which mathematical and geometrical concepts emerge and work together. It is beneficial in several ways to preserve this structure when creating a finite element software environment: First, the accessibility of the software framework for users with mainly mathematical background knowledge is improved; having worked out a FE formulation in mathematical terms, the translation into computer code is simplified if the same concepts are used to build up the code as for the analytical formulation. Second, software artifacts representing the general concepts of finite element algorithms have a high potential of reuse. A typical one-off FE application code tries to use as many simplifications as the structure of the problem at hand allows. Consequently, the resulting code does not reflect the abstract mathematical concepts any more. The latter, however, are the general building blocks for many formulations and hence providing them increases the re-usability. The clear-cut entities in FEMs can be preserved well by an object-oriented (OO) programming model.

6.3.1 Object-oriented development and programming

Object-oriented design (OOD) emphasizes analysing the problem domain in terms of “objects”, building class hierarchies exploiting inheritance and polymorphism, increasing modularity by encapsulating data, and making objects communicate with each other through messages (Barton and Nackman, 1994; Wikipedia, 2007). These notions are (sometimes incompletely) provided by computer languages with support for object-oriented programming (OOP). To aid the modeling and design process, earlier techniques by Rumbaugh et al. (1991) and Booch (1994) have been combined in the Unified Modeling Language (UML), “a visual language for specifying, constructing, and documenting the artifacts of systems" (Object Management Group, 2007). In many fields, the combination of OOD and OOP has superseded the traditional procedural programming paradigm.

A disadvantage of the object-oriented approach lies in the runtime overhead that may be incurred by the introduction of higher-level software constructs. Representing the general concepts of FEMs with an object-oriented language reduces the possibilities to apply structural, problem-dependent optimization and introduces overhead for the object handling. However this effect on the runtime has to be contrasted with the greater ease of familiarization and development with the software framework: The time savings thanks to faster application development compensate for the computational overhead.

Software design patterns

In object-oriented programming, the focus of the software generation task shifts from coding to designing, i.e., analyzing the problem domain and working out the concepts and entities and their interactions. Finally, these are implemented as software artifacts. Even more than the reuse of code, the reutilization of design can save considerable effort. This realization is at the heart of *design patterns*, i.e., "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context" (Gamma et al., 1994).

Such patterns are used in many instances in hpGEM, some of which will be described in the sequel. While patterns constitute elegant reusable solutions and their scope would allow an even broader application, on various occasions their usage has been avoided. The reason is the dependence of many patterns on polymorphism and inheritance (with abstract bases). The consequence, namely indirect (virtual) function dispatch, adds a level of indirection to every member function call. In computing-intensive applications, such as scientific computing, the additional overhead, e.g. for very frequently called (innermost loop) functions, may be prohibitive. For example, hpGEM does not provide polymorphic `Element` or `Face` classes (the latter might be used to implement different boundary conditions) to avoid the extra complexity. In such circumstances, patterns that could help to reuse available solutions (e.g., the patterns adapter and decorator) are not adopted. In this case, either the available solution has to fit directly or it must be re-implemented to avoid an unnecessary performance degradation.

Decisions about these issues are partly subjective; at larger scale, the same structural patterns may be very useful, e.g. to integrate several (libraries of) equation solvers in a consistent way. Also, there are problems which inherently have a structure that is modeled by one of the patterns and cannot be simplified. For example an implementation of meshes with local refinement might exploit the composite pattern (at least to achieve part of the necessary functionality), and the basic problem cannot be solved with a completely different approach because the problem just has the structure and complexity of a composite.

6.3.2 Choice of the programming language

Having chosen the object-oriented development model, the programming language is the next item to select. In recent years, with the growing acceptance of object-oriented techniques also in the scientific programming world, C++ has been the language of choice for many developments. Reasons that suggest committing to this language include:

- C++ not only is a full-fledged object-oriented language, but also supports programming in procedural style and with elementary constructs, like loops. The latter fact is particularly beneficial for scientific computing where the nature and amount of data requires such constructs and their execution at close to machine speed. Furthermore, the overhead to get users without experience in object-oriented programming started with hpGEM is reduced by the availability of these more traditional concepts.

- C++ enforces strong type checking, which leads to an unambiguous API and enables one to find many logical errors at compile time.

- Aspects of generic programming are included in C++, most notably in the form of templates. In the scientific computing context, templates are also used to improve runtime performance, cf. (Veldhuizen, 1995), for classical examples. The usage of templates is reconsidered at the end of this section.

- C++ is a widely available, standardized, well-known, current language. These properties make the language a solid foundation to build on: standard-compliant code is guaranteed to run on a wide range of platforms with several available compilers. Its widespread use, not only in scientific computing, means that support for C++ will not cease on the foreseeable timescale of the project, that there are qualified scientists to contribute to the project, and vice versa that expertise gained in the project is of wider relevance as well. Last but not least, C++ is one of the most supported languages when it comes to techniques and tools for software engineering, which can significantly contribute to the success of a project.

A possible alternative in the language decision would have been Fortran 90/95, which is still widely used for scientific computing. However, Fortran cannot match with C++ on the above list of qualities. In the versions for which compilers are currently available, Fortran incompletely supports object-oriented programming and lacks some more evolved object-oriented features of C++ (e.g., multiple inheritance). The basic data type of Fortran—on which it admittedly outperforms most other languages—is the array, but less structured types as they are for example readily provided by the C++-Standard Template Library (STL) containers are not offered, and generic programming support as through C++-templates is not present in the language. Obviously it would not be necessary to limit oneself to a single programming language, as linking object files generated from several languages is a viable option. While it is possible to use external software packages programmed in other languages, the development of hpGEM itself takes place in C++ only, as mixed language programming would potentially decrease portability and increase the complexity for the developers.

Of course, choosing C++ means making a compromise that includes disadvantages, too. The absence of many services and data structures (like arrays) from the language is obstructive and has to be compensated by libraries. With the advent of relevant libraries the functional range matches up with other languages, but because of the inclusion model for header files there is an increase in compilation time overhead. The overhead is often subject of complaint when the code relies heavily on templates. As mentioned above, templates are frequently used not only to achieve generality through static polymorphism but also to improve performance through compile time code expansion. While the C++ standard includes explicit instantiation as a means to avoid the overhead of the inclusion model for template definitions, this can most frequently not be used for the purposes above because the template arguments are not determined by the library but rather by the user's code. As a simple example, the class template

```
template <DimType dim> class PhysSpacePoint<dim>;
```

would be suitable for explicit instantiation since the parameter `dim`, which gives the dimension of the physical problem space, is (practically) restricted to the range $1, \dots, 4$. On the other hand, a construct like

```
template <class UserData> class DataOnElements;
```

cannot be instantiated at the build time of the hpGEM library, as its argument, namely the class that includes all data that is stored for each element, is problem dependent and hence given by the user. The latter case is, unfortunately, far more typical of the code of the hpGEM framework, which relies heavily on templates. Furthermore, Vandevoorde and Josuttis (2003) point out their experience from large projects, which suggests that explicit instantiation becomes hard to manage. Hence, up to now, explicit instantiation has not been used in the hpGEM framework; the size of the code base and the currently available computing power keep the compilation times in a tolerable range.

An aspect that should be mentioned—though it concerns the object-oriented approach rather than the choice of C++ as the programming language—is that thinking in an object-oriented way is a skill that requires learning and experience. It often takes considerable time to adapt new users to this thinking and programming style, as it is fundamentally different from traditional programming paradigms frequently taught to students—even if C++ is used for that purpose. Hence the education of students and researchers with regard to C++, object-oriented software development, and hpGEM itself plays an important role for the future prospects of the framework.

6.4 Problem domain setup

The hpGEM framework is organized in several C++-namespaces which contain software artifacts that belong together. This section gives a bird's-eye view of the partitioning. For each namespace, a few highlights are presented in more detail. Additional information can be found in the in-code documentation of the hpGEM source (hpGEM).

6.4.1 Namespace `Base`

Although the namespaces are not layered, `Base` contains functions and data structures that are used by entities from all namespaces. An example are implementations of fixed- and variable-sized arrays. `Base` provides associated functions, for instance to compute several norms and the standard determinant, too. These components are also accessible to the user. More complex tasks implemented within this namespace include the projection of initial data on the function space defined by a FE mesh and the basis functions. Two data structures of particular importance to almost any user code are described next.

User data storage: `DataOnElements` and `DataOnFaces`

The class templates

```
template <class DataType> class DataOnElements;
template <class DataType> class DataOnFaces;
```

provide containers that accommodate (the user's) data sets for each element and face. The data necessary on each element usually includes at least the expansion coefficients of the solution and/or the residual with respect to a basis. Additionally, other parameters can be stored, for instance method-specific geometric quantities, intermediate results that are needed again and would be costly to recompute, or—in a multifluid simulation—the information which fluid is occupying the element's space domain. It is part of the philosophy of hpGEM to leave the definition of this class to the user. If specific framework services are used, e.g., the projection of the initial condition on a given basis, then parts of the user class interface have to adhere to rules set out by the package design, for example by offering specific member functions or type declarations (**`typedef`**s). The element and face data containers are also used internally by hpGEM, namely to cache some geometry information that would otherwise have to be recomputed frequently, cf. Section 6.4.4.

Basis functions and expansions

As there is no global continuity requirement in DG FEMs, the reference space basis functions may be the same for all reference geometries. This is exploited by the tools that hpGEM provides for defining and evaluating (DG) basis functions and expansions in terms of them. A sequence of basis functions $\hat{\psi}_j$ (for $d = 3$) is defined as a sequence of class template specializations in terms of an integer argument:[1]

```
const DimType dim = 3;          // (space-time) dimension
typedef RefSpacePoint<dim> RSP; // type abbreviation

template <unsigned int i> struct PsiHat;

template <> struct PsiHat<0>{    // constant basis function
  enum { LastFunctionIndex = 4 };
  static NumType eval(const RSP&){ return 1.; }
  static NumType evalDeriv0(const RSP&){ return 0.; };
  static NumType evalDeriv1(const RSP&){ return 0.; };
  static NumType evalDeriv2(const RSP&){ return 0.; };
};

template <> struct PsiHat<1>{    // 1st linear basis function
  static inline NumType eval(const RSP& p){ return p[0]; }
  static inline NumType evalDeriv0(const RSP&){ return 1.; };
  static inline NumType evalDeriv1(const RSP&){ return 0.; };
  static inline NumType evalDeriv2(const RSP&){ return 0.; };
};
// Etc., implement function and derivatives for indices up to
// four - as indicated by LastFunctionIndex.
```

The following declarations make this basis known to hpGEM:

```
typedef Expansion<dim, PsiHat> ExpansionDefs;         // (1)
typedef ExpansionDefs::ServerType BasisExpServer;     // (2)
BasisExpServer::InitializeFunctionPointerArrays();    // (3)
typedef ExpansionDefs::ExpansionType ExpansionType;  // (4)
```

(1) By fixing the problem dimension and the basis functions, `ExpansionDefs` yields a shortcut for the expansion-related definitions that follow.

(2) The type `BasisExpServer` names the class that internally evaluates expansions.

(3) The evaluation of expansions is achieved by different functions, depending on the current length of the expansion, i.e., the number of basis functions used. These evaluation functions are generated by template meta-programming, and a pointer to each function is stored internally by the `BasisExpServer`. The initialization of the pointer fields has to be induced with this statement.

[1] In the code fragments included in this chapter, full scoping is usually avoided, e.g. `Geometry::DimType dim` is abbreviated as `DimType dim` to keep the listings simple.

(4) Another abbreviation names the type of the actual expansions in terms of the basis `PsiHat`. Subsequently, expansion variables can be defined in terms of the basis functions as

```
ExpansionType expansion;
```

Objects of type `ExpansionType` can be queried directly at a `RefSpacePoint<dim> pRef` (cf. namespace `Geometry`) in the reference element, as

```
NumType valueOfExpansion;
expansion(pRef, valueOfExpansion);
```

The advantage of `ExpansionType` is that state evaluation is carried out internally and efficiently: As each instance of `ExpansionType` carries information about the number of currently used basis functions, the address of the corresponding evaluation function for this expansion length is retrieved, and with one call to that function the complete expansion is evaluated. In particular this avoids further function calls to the individual `PsiHat` member functions. The instantiation of the evaluation functions relies on template meta-programming techniques (Veldhuizen, 1995), which make it possible to expand the linear combination at compile time, making it available to the optimization tools of the compiler, too.

However, the machinery explained above depends on the element-independence of the basis functions. This is special to DG methods, and even for these it is not universal: Besides the reference element DG basis functions $\hat{\psi}_j$, van der Vegt and van der Ven (2002b), for example, use a second set of basis functions

$$\hat{\psi}'_j(t,\bar{x}) = \begin{cases} 1\,, & \text{for } i = 0, \\ \hat{\psi}_j(t,\bar{x}) - \frac{1}{|\bar{\mathcal{K}}_e^{n,-}|} \int_{\bar{\mathcal{K}}_e^{n,-}} \hat{\psi}_j((G_e^n)^{-1}(t^n,\bar{x}'))\,\mathrm{d}\bar{\mathcal{K}}\,, & \text{otherwise,} \end{cases}$$

to separate the mean and slope on the physical element, which is essential for both their stabilization and multigrid methods. Cockburn et al. (2000) describe a hierarchical tensor-type basis with different basis functions for the different element reference geometries. Finally, of course, for continuous Galerkin methods, the basis functions—though not element-dependent—have to be matched with the global degrees of freedom, thus requiring extra information. These examples testify that hpGEM will need another model for bases. Beyond the features needed for the previous three examples, such a model should allow caching the function and derivative values at a set of points in the (reference) element. Because of the complications involved in this undertaking (several rules with different integration points may be used, different bases on different elements, integration on both faces and elements, etc.), it has not been tackled yet.

6.4.2 Namespace `Geometry`

In Chapter 3, several geometric concepts were introduced that occur in FE formulations. Many of them have direct counterparts in hpGEM, as will be shown next. In particular the `Elements`,[2] their `ReferenceGeometry`, which simplifies the definition of basis functions and the integration with quadrature rules, cf. Appendix B, and the `Faces`. More fundamentally, the FE `Mesh` is defined by the positions of a set of nodes in physical space and the connectivity of these nodes to yield elements and faces. The setup of such a mesh is a complex task, but this step is hardly connected to the equations solved or the FE method. hpGEM encapsulates mesh setup as a complete service provided by the framework. With a few commands that concern the origin and type, the user obtains the mesh and the framework is ready to work with it. Similar services are provided by other FE packages, as described in Section 6.2, but hpGEM differs from them in that it takes dimension-independent programming to the full, even on unstructured meshes. The commands

```
Mesh<dim> theMesh;
CentaurMeshFileReader<dim> Reader("mesh.hyb", theMesh);
```

read the file `mesh.hyb` generated by the Centaur mesh generator (Centaursoft, 2005), in particular unstructured meshes with mixtures of the possible element geometries: triangles and quadrilaterals when `dim` is two, and tetrahedra, pyramids, triangular prisms, and hexahedra in three-dimensional meshes. The elements and faces, including the appropriate geometry description—consisting of a reference geometry, the physical coordinates and a corresponding mapping—are generated automatically by suitable factory methods, cf. (Gamma et al., 1994). The boundary faces are typically provided by the mesh generator. On the other hand, the internal faces, which are required by DG FEMs, are not. Hence hpGEM provides a general algorithm to generate them. The work flow of the generation of the FE mesh representation in hpGEM is illustrated in Figure 6.1 on the next page.

All information is collected in a `Mesh` object, whose most prominent task is to provide iterator-like access to the element and face containers. Since most meshes in practice are unstructured,[3] no access other than iterators is provided. Consequently, a rectangular 8×8 mesh on the unit square generated with an instance of the hpGEM `RectangularMeshGenerator` class by

```
PhysSpacePoint<2> p1; p1[0] = p1[1] = 0.; // lower left
PhysSpacePoint<2> p2; p2[0] = p2[1] = 1.; // upper right
unsigned int nrOfEl[] = { 8, 8 };         // 8x8 elements
RectangularMeshGenerator<2> rmg(p1, p2, nrOfEl, theMesh,...);
```

[2] In this context, 'element' refers only to the part of the domain, not the triptych as, e.g., in the definition by Ciarlet (1978).

[3] Meshes are considered unstructured if there is no (global) rule to identify neighbors, i.e., neighborship information has to be stored locally. For DG algorithms this typically happens on the face.

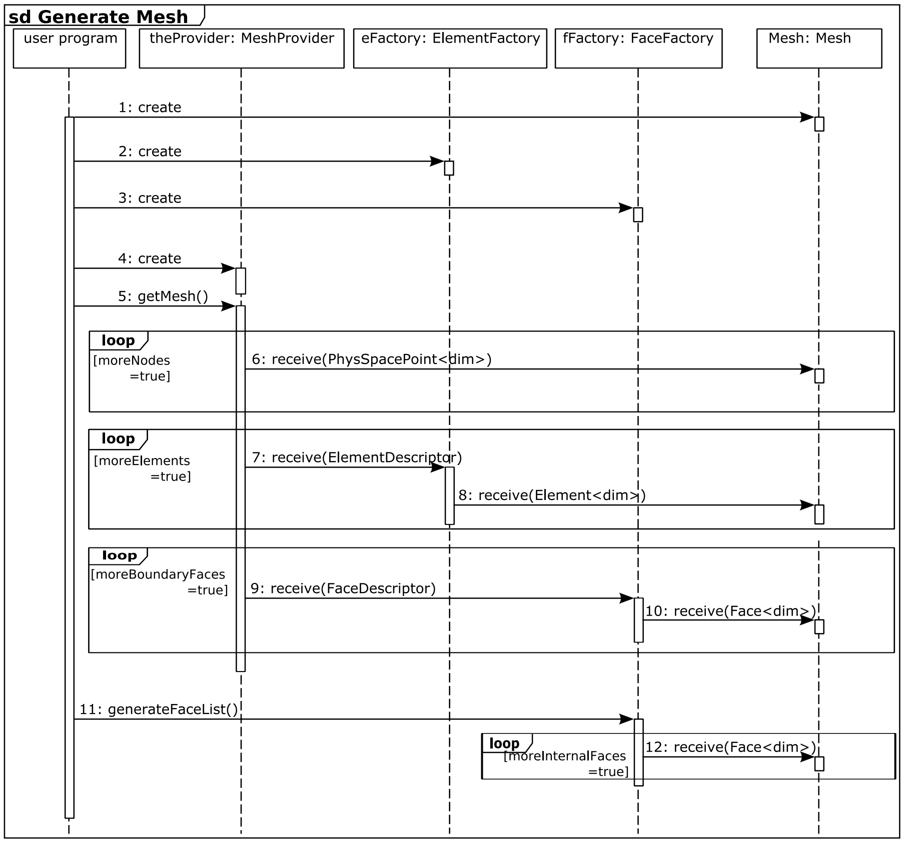


Figure 6.1: UML sequence diagram of the setup of a mesh. In the example of generating a rectangular mesh, the user-induced actions in the diagram correspond to the following program code:

```
Mesh<2> mesh;                                          // (1)
ElementFactory<2> eFactory(mesh);                      // (2)
FaceFactory<2>    fFactory(mesh);                      // (3)
RectangularMeshGenerator<2> rmg(..., mesh,             // (4)
                                  eFactory, fFactory);
rmg.getMesh();                                         // (5)
fFactory.generateFaceList();                           // (11)
```

(Some constructor arguments have been omitted in the declaration of the `RectangularMeshGenerator<2>`, see the example on page 115.) The internal function calls deliver the physical space nodes (6) from the mesh provider to the mesh data structure. The nodes are numbered and based on this enumeration, each instance of `ElementDescriptor` contains the ordered sequence of global node numbers defining one element (7). The element is produced by the element factory with its reference and physical geometry and the mapping between the two, and sent to the mesh (8). The boundary faces are normally delivered by the mesh generator, too, and a similar procedure applies (9,10). The internal faces are generated by hpGEM after a call from the user code (11,12).

does *not* have access by element coordinates `(i,j)`. The generality of the approach with iterators is preferred: By exchanging the few code lines shown above for generating a rectangular mesh with those used earlier for reading a file from an (unstructured) mesh generator, the iterator-based program can work on any mesh in any dimension, while the version with indices or other direct access cannot.

Further services implemented by hpGEM as mesh processing steps are methods to

1. apply transformations to the nodes of a mesh,
2. make a mesh periodic,
3. add a (time) dimension to the given space coordinates.

The first two enable to adapt meshes to special geometries, e.g. a mesh on a cube can be used to discretize a torus. The last feature is needed for space-time DG methods, in which the space and time discretization are carried out in one step (rather than discretizing space and time separately), see Chapter 5 and (van der Vegt and van der Ven, 2002b). The communication between all mesh generation-related objects takes place through abstract interfaces in a sender-receiver pattern, so that the above tools may be combined in a chain.

To conclude, the philosophy of hpGEM regarding the geometry setup is to provide front-ends to mesh sources, in particular (commercial) mesh generators, encapsulate the geometry information generation, and provide simple but universal means for unstructured mesh usage.

6.4.3 Namespace `GlobalAssembly`

As has been mentioned in Section 6.3.2, the hpGEM framework does not dictate the number or type of variables per element and not their mathematical connection either. Consequently, the concrete handling of the data—like initialization, reading it out, or updating—has to be carried out by the user's code. Nevertheless, many algorithms can be encapsulated in and provided by hpGEM, if they are designed around such handling methods, e.g. by using the *template method* pattern by Gamma et al. (1994).

For example, a typical step in a finite element computation is the assembly of the matrix of a linear system of equations. This step is usually subdivided into computing a contribution locally on each element and subsequently sorting these local contributions into the global matrix.[4] In this case, the computation of the local matrix has to be done by the user (using other tools provided by hpGEM), but the sorting into the global matrix can be left to the routines provided in namespace `GlobalAssembly`, as the necessary links to element or degree of freedom bookkeeping are anyway handled by the geometry classes.

[4]The first of these steps is completely local and easily parallelizable. The second, in contrast, is not, as—depending on the FEM—the mapping to global matrix entries is not injective. Parallel programming paradigms like semaphores, however, allow to execute also this part of the assembly in parallel.

Some of the tools provided by the GlobalAssembly components can be demonstrated by example of the discontinuous Galerkin discretization of the Poisson equation $-\nabla^2\phi = f$ by van der Vegt and Tomar (2005). In this method, the assembly of a global linear equation system is necessary because the discretization uses lifting operators on the faces to discretize the Laplace operator. Assuming that an instance mesh of type Mesh<dim> and a DataOnElements<UserData> container elData have been defined, the program sequence that represents the assembly is

```
const unsigned int nDOF = countGlobalNrOfDOF(mesh, cDOF);// (1)
GlobMat A(nDOF, nDOF); GlobVec x(nDOF), b(nDOF);         // (2)
GlobalLapackMatrixSorter<ElementIDType> sorter(A, x, b); // (3)
assembleElementContributions(mesh,                       // (4)
                             CalcElContrib<dim>(elData),
                             sorter);
assembleFaceContributions(mesh,                          // (5)
                          CalcFaceContrib<dim>(UserData),
                          sorter);
lapack::gesv(A, b); x=b;                                 // (6)
sortSolutionBack(mesh,                                   // (7)
                 Solution2ElementData<dim>(UserData),
                 sorter);
```

The call in line (1) of the example determines the number of degrees of freedom of the discrete problem. It requires a function or functor cDOF that can evaluate the number of degrees of freedom on a single element. This functionality must be implemented by the user; it is rather trivial, so that a sample implementation is omitted here. Based on the number of degrees of freedom in the linear system, the matrix A, the right hand side vector b, and the solution vector x are declared in (2). These linear system components are passed to hpGEM's GlobalLapackMatrixSorter (3), which provides a front-end to the global system and internally handles the mapping from local degrees of freedom per element to the global matrix entries. The calls to assembleElementContributions and assembleFaceContributions in (4) and (5), respectively, initiate the actual assembly procedure. They rely on the user's implementation of the computation of the local contribution per element and face in the functors CalcElContrib<dim> and CalcFace-Contrib<dim>. An example for the element term will be sketched after the completion of the discussion of the above code fragment. In (6), the global system of equations A x = b is solved with the help of the Lapack (Anderson et al., 1999) routine GESV, accessed through the boost-bindings (boost C++ libraries). Finally, the solution has to be mapped back from the mere vector entries in x to the expansion coefficients of the finite element data, which happens in (7).

The example documents well how the framework code and the user's implementations of algorithm- and equation-specific parts work together. To complete the discussion of the global assembly, the explanation of one of the user-implemented parts is added. For the

Poisson equation, the right hand side vector contains the L^2-inner product of the function f with each of the basis functions $\hat{\psi}_j$. The computation of this contribution is one part of the functor `CalcElContrib<dim>`, whose **`operator()`** is called by the hpGEM function `assembleElementContributions`:

```
void operator()(const Element<dim>& el,
   ElementLocalSystemAcceptor<ElementIDType>& eAcceptor)
{
  LocalRHSType r; r.resize(nDOFlocal); r.clear();     // (1)
  for (int j = 0; j < nDOFlocal; ++j)
    integrateOverElement(..., r(j));                   // (2)
  eAcceptor.assembleElementLocalRHS(el.id(), r, add); // (3)

  // Element contribution to global matrix A computed here
  // and passed to eAcceptor.assembleElementLocalMatrix().
}
```

The operator takes a reference `el` to the `Element<dim>` instance for which the contribution is expected and an `ElementLocalSystemAcceptor`, which organizes the local contributions into a global system (in the previous code example on the facing page this role is fulfilled by the `GlobalLapackMatrixSorter`). The vector `r` is declared in (1) with as many components as basis functions are used in the local expansion on the element (which may vary between elements but is available through the reference to `el`). It stores the results of the integration of the product of f and the basis function $\hat{\psi}_j$ in (2). The call syntax for the integration is described on page 123 in the section about the integration framework of hpGEM, and thus omitted in the example. In (3), the user code in turn invokes a member function of the `ElementLocalSystemAcceptor` to add the local contribution from `r` to the right hand side vector of the global system. The collaboration between the different entities and the call sequence of the element assembly are also documented in Figure 6.2 on the next page.

Notably, the above code neither depends on the dimension nor on the reference geometry of the element. This matches with the independence of the DG discretization of both properties. In (Pesch et al., 2007), numerical examples are given based on a two-dimensional mesh with a mixture of triangular and quadrilateral elements and for a cubic mesh in three dimensions. Both are produced with the same implementation, from which the above code fragments originate.

6.4.4 Namespace `Integration`

Due to the nature of finite element methods, their fine-grain building blocks are integrals over elements and faces, cf. Chapter 5. Integration is linked to several other concept areas, e.g. the different reference geometries in a mesh (like triangles and quadrilaterals in two dimensions, cf. Table 3.1 on page 41), algorithm-specific details (like the actual choice of

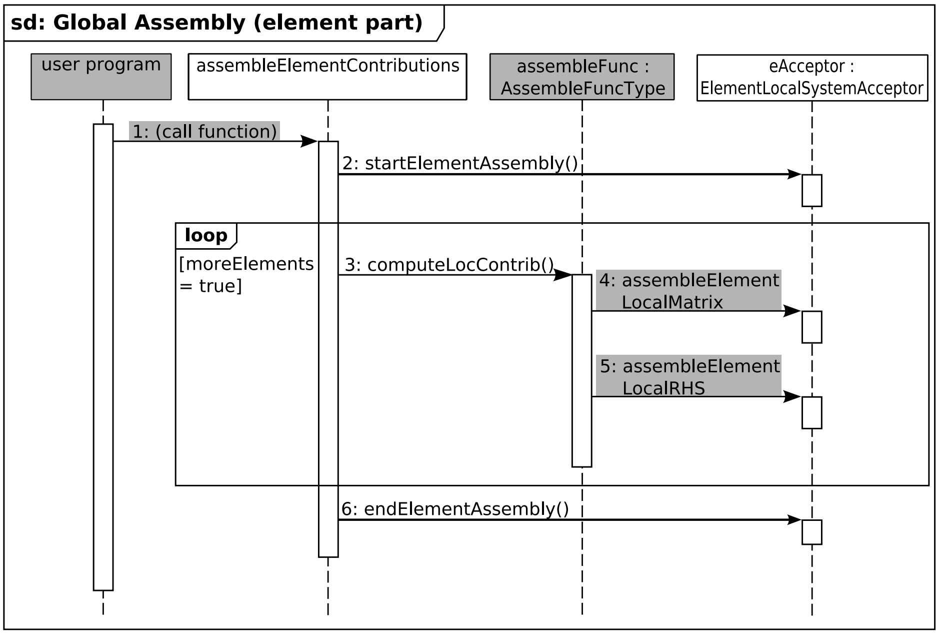
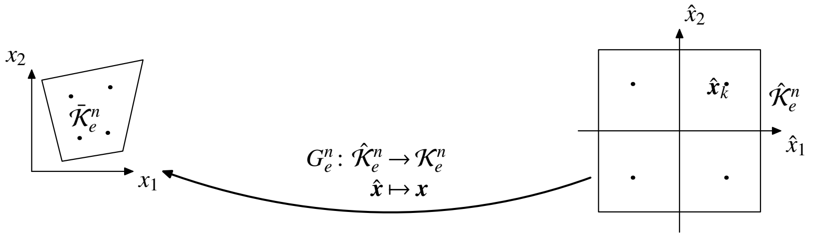

sd: Global Assembly (element part)
user program
assembleElementContributions
assembleFunc : AssembleFuncType
eAcceptor : ElementLocalSystemAcceptor
1: (call function)
2: startElementAssembly()
loop
[moreElements = true]
3: computeLocContrib()
4: assembleElement LocalMatrix
5: assembleElement LocalRHS
6: endElementAssembly()

Figure 6.2: UML sequence diagram of the element part of the global assembly process. The figure shows how the user's code and the framework services intertwine. The user-provided entities and calls are shaded in gray. On being called by the user code (1), the hpGEM function `assembleElementContributions` takes care that the destination data structure is notified of the start and end of the assembly operation (2,6). The main task is the computation of the element local system contributions, which is delegated to the user's function/functor (3). From there, the user passes the local matrix (4) and right hand side (5) blocks to the data structure that is responsible for the mapping from local to global degrees of freedom and for administrating the global system of equations. See also the code fragment on the previous page.

an integration rule depending on various factors specific to the method), and the variety of mathematical objects to be integrated (e.g., scalars, vector fields). Therefore integration has been one of the complicated design challenges, with the flexibility of the integration interface being a prerequisite for usability in various contexts.

While some of the integrals may be computable exactly (when the integrand is of simple analytical form), usually they are evaluated by numerical quadrature. In hpGEM, integration is based on the application of quadrature rules, i.e., a weighted sum of integrand evaluations at a set of points on the reference shape is computed, cf. Figure 6.3 on the next page. The function call to compute an element integral takes a reference to the element over which to integrate, (optionally) the Gauß integration rule to be used, the function to



x_2
$\mathcal{K}_e^n$
x_1
$G_e^n\colon \hat{\mathcal{K}}_e^n \to \mathcal{K}_e^n$
$\hat{\boldsymbol{x}} \mapsto \boldsymbol{x}$
$\hat{x}_2$
$\hat{\boldsymbol{x}}_k$
$\hat{\mathcal{K}}_e^n$
$\hat{x}_1$

Figure 6.3: The integral of a function f over the element $\mathcal{K}_e^n$ is transformed to the appropriate reference element $\hat{\mathcal{K}}_e^n$ through the mapping G_e^n. For the reference geometry, numerical integration rules are available as a weighted sum of function values at a set of points $\hat{\boldsymbol{x}}_k$ with weights α_k. The transformations are taken care of by the integration framework of hpGEM, and so is the computation of the Jacobian of the mapping.

integrate and a reference for the result storage:

```
integrateOverElement(element, quadRule, integrand, result);
```

The transformation between reference and physical space takes place automatically by also evaluating the Jacobian of the transformation. Quadrature rules of order up to at least seven for all supported reference geometries (cf. Section 3.2) have been implemented based on (Stroud, 1971). Since every integration rule is for a fixed dimension, they can be compiled once into a library, unlike C++ class templates, whose definition has to be available at compile time of the application. In fact, also the code for the integration rules is generated only for the preparation of the library. The code generation uses a simple description format, in which, for example, the first order integration rule on the $[-1;1]$ reference line is given as

```
GaussRuleFromPoints C1d(
    "Cn1_1_1",              // name; cf. Stroud (1971)
    "centroid formula 1d", // explanation
    "ReferenceLine",        // reference geometry
    1);                     // order
C1d.addIntegrationPoint("2.0  [ 0.0 ]"); // weight&coordinate
```

Product rules can be constructed simply by giving the two lower-dimensional input rules:

```
GaussRuleProduct Cn2_1_1(
    "Cn2_1_1",              // name; cf. Stroud (1971)
    "centroid formula 2d", // explanation
    "ReferenceSquare",      // reference geometry
    &Cn1_1_1,               // rule for first dimension
    &Cn1_1_1,               // rule in other dimension(s)
    1);                     // order
```

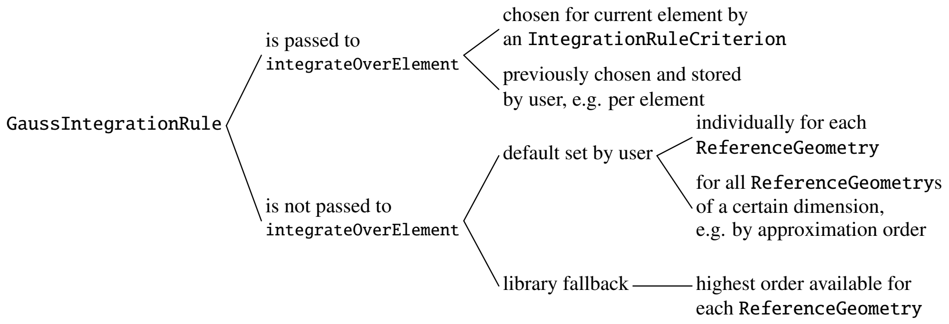

GaussIntegrationRule
is passed to integrateOverElement
chosen for current element by an IntegrationRuleCriterion
previously chosen and stored by user, e.g. per element
is not passed to integrateOverElement
default set by user
individually for each ReferenceGeometry
for all ReferenceGeometrys of a certain dimension, e.g. by approximation order
library fallback
highest order available for each ReferenceGeometry

Figure 6.4: Different ways to choose the numerical integration rule for computing an integral.

The code generation makes it easy to extend the set of integration rules without having to understand the details of how the rules are implemented. This includes that each rule is a singleton (Gamma et al., 1994), which guarantees that there exists only one instance of each rule in a program. Admittedly deriving separate classes for different rules violates the principle that virtual functions should be reserved for variation in behavior, as opposed to variation in value (Cargill, 1992), but in this case preference was given to the separate derivation of the rules from the base class, which also gives each rule a distinct class name for selection by the user.

All rules register themselves automatically with the reference geometry they belong to. That way, each reference geometry knows which rules are available to integrate on it and the user can apply selection criteria to find an appropriate rule. For that purpose, different possibilities exist, see Figure 6.4. For example, a choice can be made based on the name of the rule or the approximation order it offers. The latter is particularly interesting as it allows to require that all integrals have to be computed with (at least) a certain order of accuracy:

```
IntegrationRuleByOrderEQ crit(5);       // Use a 5th order rule
setElementIntegrationOrder<dim>(crit); // for element integrals.
```

On the other hand, the user can also decide on an element-by-element basis which rule to use; this feature is required for FEMs with p-refinement, i.e., the approximation order of the FE space is allowed to vary on different elements.

A crucial aspect of the integration routine is that the types to be integrated and the result type are templatized. The condition on the integrand function is that it maps from a reference geometry to some linear space S, i.e., $\varphi : \hat{\mathcal{K}} \to S$. In terms of C++ that means the function evaluation $s = \varphi(\hat{x})$ for $s \in S$ and at the reference shape coordinate $\hat{x} \in \hat{\mathcal{K}}$ can be carried out in the function call syntax as `phi(const RefSpacePoint<dim>& x, S& s)`, i.e., it is either a function with this prototype or a functor with a corresponding `operator()`. Through a traits system (Myers, 1995) the result type of the integral is deduced from the

Table 6.1: Runtime comparison with and without saving of the face normal vectors. The comparison uses entropy variables V or conserved variables U and an Euler equation test case.

		runtime [s]	
		V	U
face geometry:	recomputed	128	76
	saved	77	41

input types. Using this system, also various products can be formed and integrated. For instance for the projection of a physical space function $f\colon \mathbb{R}^d \to \mathbb{R},\ x \mapsto f(x)$, onto a set of element local basis functions $\{\hat{\psi}_j,\ j = 1, \ldots, n\}$ the integral $r_j = \int_{\mathcal{K}} f(x)\, \hat{\psi}_j(G^{-1}(x))\, \mathrm{d}\mathcal{K} = \int_{\hat{\mathcal{K}}} f(G(\hat{x}))\, \hat{\psi}_j(\hat{x})\, |\mathrm{Jac}_{\hat{x}} G|\ \mathrm{d}\hat{\mathcal{K}}$ on the element $\mathcal{K}$ is computed as

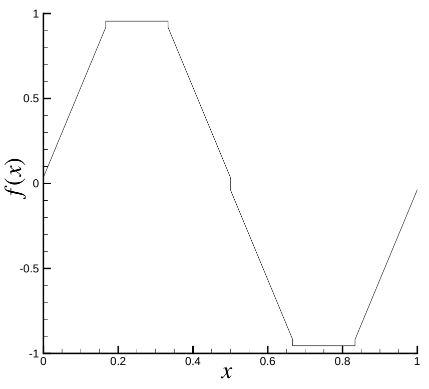
```
integrateOverElement(K, integrationRule,
    scalarProduct<d>(K.transform2RefElement(f), psi(j)),
    b(j));
```

where `integrationRule` denotes a pointer to an explicitly chosen integration rule. The `transform2RefElement(f)` command wraps the user's function, which expects physical space coordinates, so that it can be queried at reference space points by the procedure `integrateOverElement`.

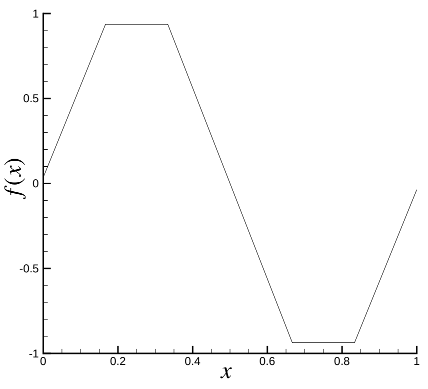
If the integrated function is not provided by hpGEM and its type is not included in the predefined traits, the user can extend these rules by template specialization. Similar functionality as described above for elements is also available for face integrals.

Saving geometry

In its first implementation, the geometry framework of hpGEM was fully general for moving meshes with space- and time-dependent element and face mappings. On the one hand, this approach serves any eventuality, but on the other hand in most cases it is overly general and unnecessarily costly: Geometry information—for example the normal vectors on the faces—are typically recomputed (i) in every Gauß point, (ii) of every face, (iii) in every (solver) iteration, and (iv) of every time step. To avoid (i) and (ii) one would have to make assumptions restricting the type of mesh that can be used: planar faces or sets of parallel faces, respectively. These assertions are restrictive and also sometimes hard to verify. In contrast, the items (iii) and (iv) are relatively easily communicated and checked, and in fact such constant-in-time meshes are common for the solution of many problems. Hence the integration framework has been extended so that it can optionally be instructed to internally cache geometry information—namely the normal vectors on the faces and the Jacobian of the element mappings—and reuse it. For the case of a constant mesh, performance results with and without storing geometry information are listed in Table 6.1. They were obtained from a typical run of the Euler solver from Chapter 5 for a fixed number of time steps. Thanks to the saved geometry computations the overall simulation executes 40% faster for entropy variables and 46% when conservation variables are used.



(a)



(b)

Figure 6.5: The function $f(x) = \sin(2\pi x)$ L^2-projected onto six equal-sized elements with $p_\mathrm{s} = 1$ DG basis functions on the interval $\Omega = [0; 1]$. In (a), the discrete function is plotted with its discontinuities (the jumps are connected by the visualization software). A different plotting routine automatically makes the projected function continuous by nodal averaging, see (b).

The example supports two findings: first, the speedup depends on the total amount of computation in the implemented algorithm, but, second, whenever the constant mesh geometry assumption is viable it should be exploited as the benefit in execution time is considerable.

6.4.5 Namespace `Output`

The results of numerical simulations frequently need to be processed for graphical presentation. hpGEM offers several output classes coupling it to Tecplot (Tecplot, Inc., 2005). Different implementations allow writing of the complete mesh and data or reduction by one dimension, which enables to write space-only output from space-time meshes. Further, the output can be written on a per-element basis, meaning that the discontinuous character of the solution is preserved, or as nodally averaged, continuous fields, cf. Figure 6.5. Also the plots in Chapter 5 have been prepared with these output facilities.

6.5 Examples from the DG Navier–Stokes implementation

Having discussed several parts of the hpGEM framework, a few points will be added which stem from the experience gained during the implementation of the discontinuous Galerkin method derived in Chapter 5 for the Euler and Navier–Stokes equations. Other applications of hpGEM are discussed in (Pesch et al., 2007), among them an elliptic problem (Poisson equation) and a method for interface tracking based on the cut-cell and level set methods.

Here, at first two examples of general programming or design principles are given to illustrate how consistency within the application code and coherence with the implemented problem can be improved. Thereafter, a larger-scale design is proposed that concerns the different flux entities in hyperbolic equations.

6.5.1 Consistent use of different variable sets

One of the advantages of C++ mentioned in Section 6.3.2 is its static type checking; the compiler has to be able to infer the type of every object so that it can be guaranteed that the operations carried out on the object are defined. Appropriate design can exploit the type checking mechanism to ensure consistency in the program. In the generalized variable approach discussed in Chapter 2 and used in the implementation of the method from Chapter 5, the problem arises that the implementation should accommodate the different variable sets, but depending on which set is used, different transformations have to be carried out. The data structure that represents a state of any of the discussed variable sets in space-time dimension d is an array of length $d + 1$. However, when a state is brought to that level, the information which variable set it is based on is lost. By setting up a distinct type for a conservative variable state as

```
template<DimType dim>
class Ustate
    : private FixedVector<NumType, dim+1>
{ public: using FixedVector<NumType, dim+1>::operator[]; };
```

and analogous declarations for V and Y states, the possibility to exploit the variable type information is restored. The simplicity of the `Ustate` class template is its strength: the absence of assignment operators to other types prevents different state types from being used inconsistently. No operations other than vector entry access are defined for `Ustate<dim>` objects, because most operations (e.g., scalar multiplication) do not make sense for such a type. Functions operating on states are type-safe by declaring their arguments in terms of the class templates, for example,[5]

```
template <DimType dim, class VariableSet> // empty default!
struct ComputeA0 {};

template <DimType dim>
struct ComputeA0<dim, Uvariables<dim> >
{  static void go(const FluidThermodynamics*,
```

[5]In the example, the partial specialization is on a type `VariableSet`. The classes that take its place, `Uvariables<dim>` etc., are extensions of the U, Y, and V state implementations and add facilities like transformations between variable sets.

```
                        const Ustate<dim>&,
                        MatrixType& mA0) { /* fill mA0 with
                                              identity matrix */ }
};

template <DimType dim>
struct ComputeA0<dim, Vvariables<dim> >
{  static void go(const FluidThermodynamics* fluidPtr,
                        const Vstate<dim>& V,
                        MatrixType& mA0) { /* compute A^0_V */ }
};

// Analogous specialization for Yvariables<dim>.
```

Each partial specialization for one of the variable sets accepts only an argument of the associated state type and computes the corresponding matrix A_0^V for the concerned variable set V. In the described way, the state type is carried into all parts of the implementation that have to deal with state values, leaving it to the C++ compiler to select the correct template specialization and to check that consistent argument types are used.

The choice which state type to use is made in the main program by a type declaration `typedef Vvariables<dim> CurrentVariableSet;` which can be exchanged easily for a different type. Subsequently, `CurrentVariableSet` is one of the template arguments for many class and function templates that implement the DG method. The consequence of the compile-time selection of the used state variable set is that choosing a different set requires the main program to be recompiled.[6] But this sacrifice is opposed by a clear improvement of expressiveness and quality of the software.

Incidentally, the same type safety effect is exploited by the distinction between `RefSpacePoint<dim>` and `PhysSpacePoint<dim>`, i.e., points with reference and physical element coordinates, respectively. While the underlying data structure is identical for both classes and the difference may appear minor, it has been very useful to distinguish the two types. In this way, the mathematical meaning of the input and output of the arguments of mappings, integration arguments, etc., is always clear and misuse can be avoided.

6.5.2 Implementation of the thermodynamical description of different media

The entropy variable formulation exploited in the previous chapters has the advantage that the incompressible limit is well-defined. This not only makes it possible to treat low Mach-number gas flows, but also fluids with different properties, including incompressible ones.

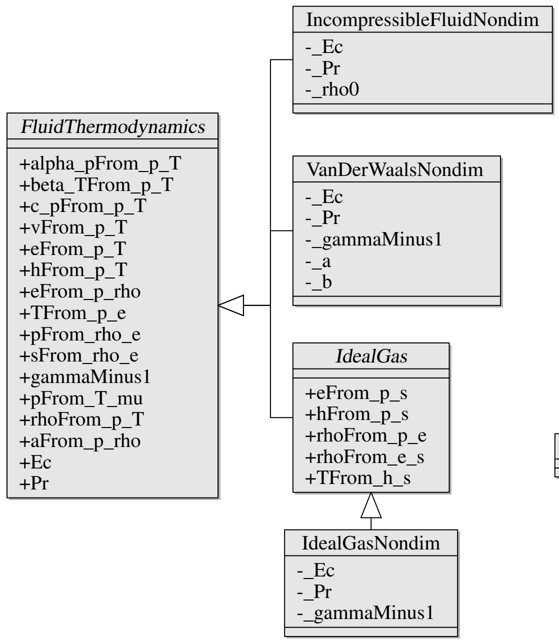
[6]Admittedly, one could instantiate all three specializations (U, V, and Y variables) in one application and thus defer the decision about the used variable set until runtime, but this is not realized in the available implementation.

An obvious but not necessarily trivial requirement for the exploitation of this capability is that the thermodynamical model equations in the implementation of the FEM have to be exchangeable. To achieve this important goal, two issues have to be settled: First, all necessary thermodynamic relations in the complete algorithm have to be cataloged. While the theoretical description derives many properties from very few basic relations, this is practically not feasible. Furthermore, the computation of boundary conditions often requires additional relations, beyond the equations for pressure and energy needed in the conservation equations. The second problem concerns the technical issue of removing the implementation of the thermodynamical relations from multiple occurrences in different code units and bundling them in a unique entity.

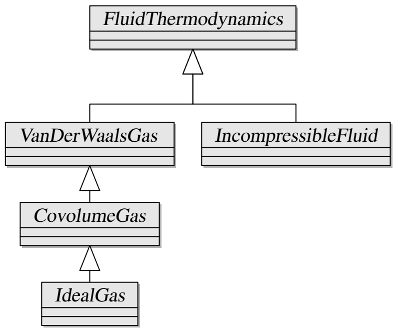
For the implementation of thermodynamical models in the framework of the Navier–Stokes solver of Chapter 5, the two problems lead to the solution documented in Figure 6.6a. A set of 16 functions is declared necessary by the abstract base class `FluidThermodynamics` for any fluid description to be used with all three variable sets introduced in Chapter 2. The qualification in the previous sentence is important: the function `pFrom_rho_e`, for instance, cannot be defined for an incompressible fluid. It is, however, not necessary to do so, because this relation is needed only by the transformations $V(U)$ and $Y(U)$. Clearly, these are not needed because the conservative variable set U cannot be employed to compute incompressible flow. Therefore, the implementation of `pFrom_rho_e` in the class `IncompressibleFluidNondim` raises an exception to warn the user about the invalid combination of problem and numerical method. An exception is, of course, a runtime consequence and this emphasizes that type-safety as in the problem discussed in the previous section cannot be achieved for the thermodynamical models. Rather, the selection of a particular model is itself a runtime task and the different implementations shown in Figure 6.6a are handled only via base class pointers `FluidThermodynamics*` in most routines.

In the course of Chapter 5, however, some parts have been added to the numerical algorithm that can work only with a restricted class of fluids. The HLLC numerical flux and the boundary conditions by Darmofal et al. (2000) are examples. As these apply only to ideal gases, there is an extra interface layer that provides a base for different implementations of this model. The derived base class `IdealGas` is itself abstract and extends the general base `FluidThermodynamics` with another five relations that are necessary only for the provision of the specialized boundary conditions. Different implementations of `IdealGas` are still possible, e.g. for constant or variable specific heats, but only one is indicated in Figure 6.6a. The inheritance tree discussed so far is the result of the continuous development within the project. It might be worth refactoring it according to Figure 6.6b to preserve more structure, and express the relations between different fluid models and algorithm components.

A final remark on the implementation of thermodynamical models in the object-oriented way of Figure 6.6 concerns the incurred runtime overhead. Typically, Navier–Stokes solvers are written for one particular fluid model, and the necessary equations of state are



(a) Abstract classes have their methods shown without arguments (these are part of the function name anyway) and the return type is a single real value. The implementations do not repeat the member functions but show only the minimal set of data values.



(b) A possible design to restore the structure of the thermodynamical models with a revised inheritance tree. All shown classes are abstract and implementations could split off ‘horizontally’ in every layer.

Figure 6.6: Simplified UML class diagrams of the current and a possible refactored inheritance structure for thermodynamical classes.

hard-wired into the code wherever necessary. Moving the evaluation of these relations into virtual member functions of the implementation of an abstract base means that one or more indirect function calls are necessary whenever a thermodynamical conversion is needed. These calls cannot be optimized by inlining as only a base class pointer is available and the class resolution cannot take place until runtime. Therefore a considerable overhead is introduced.[7] Obviously, the extra cost is incurred independent of the fluid model used. Consequently, even if a single specific thermodynamical model is used, juxtaposing runtimes of the general framework with a specialized code for the same model is comparing apples and oranges. Effectively, the described overhead is the price for being able to consider a much broader class of problems (and ultimately multifluid problems)

[7]To give an idea of the order of magnitude: for a three-dimensional space-time mesh of $10 \times 10 \times 1$ elements with only one degree of freedom per element in a single pseudo-time step with entropy variables, the Euler equations require $94,800$ calls to member functions of `IdealGasNondim`, and the viscous terms another $48,000$.

with a single numerical method and implementation.

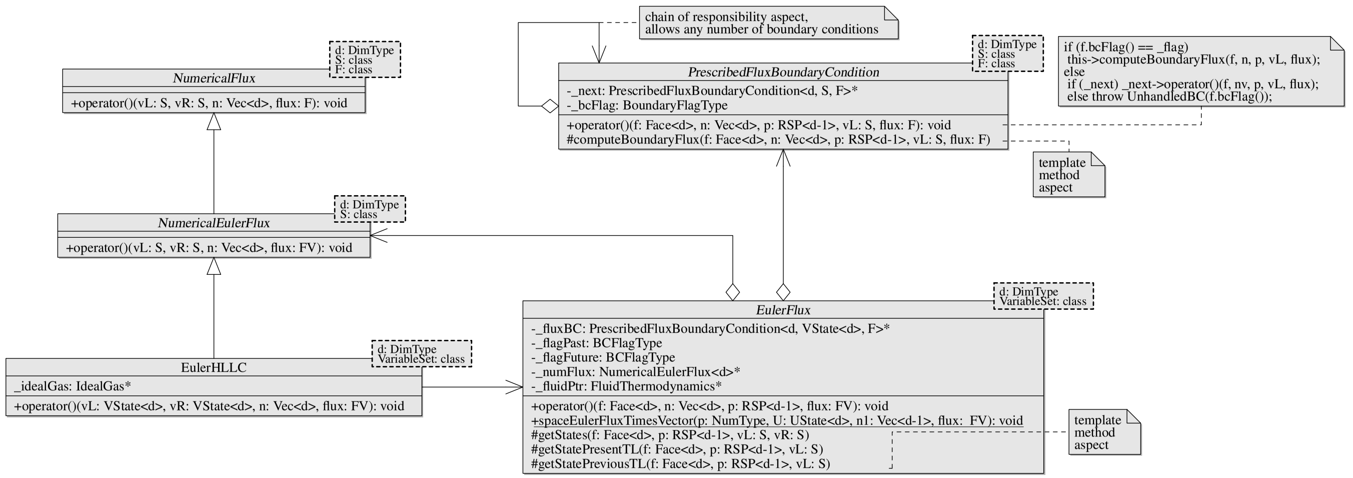
6.5.3 Fluxes for hyperbolic equations

Apart from the thermodynamical model, another entity that is needed in the weak form of the Navier–Stokes equations described in Chapter 5 is the (Euler) flux function F^{e} and its numerical counterpart $\hat{F}^{\mathrm{e}}$. The original flux F^{e} occurs in an element integral with the integrand $\nabla\hat{\psi}_j \cdot F^{\mathrm{e}}$, which is typical for conservation laws. Further, the Euler flux has a unique functional form (e.g., based on conservative variables and pressure) and thus requires a once-only implementation. By contrast, different numerical flux functions may be used. When different sets of variables are allowed, the computation of the flux may require an initial transformation to the set of quantities by which the flux is defined. Therefore, for the evaluation of the flux the variable type has to be known, while it is not needed for the definition of the flux based on a state of a specific type. To comply with these specifications the design in Figure 6.7 on the next page is proposed.

The abstract base class `NumericalFlux`[8] is derived from to fix the type of the flux result. For the Euler flux, this is a `(dim+1)`-dimensional vector, with the space-time dimension `dim`. For the current purposes, the type of the state input to `NumericalEulerFlux` is still left unspecified. This has to change in an implementation, for example in `EulerHLLC`, which must be able to compute flux values and thus be able to obtain the variable types used in its formal definition. The templatization discussed in Section 6.5.1 provides the possibility to work with a specific set of quantities without having to subscribe exclusively to one of the state types U, Y, or V. Hence the template parameter `VariableSet` selects one of the three at compile time. Polymorphism allows different numerical fluxes to be used through the `NumericalEulerFlux`-interface. It is up to the implementations of numerical fluxes to assure that their use is valid in the given context. This concerns especially the combination with a specific fluid model. For example the HLLC flux checks that the `FluidThermodynamics`-instance to which it is provided access is an `IdealGas` implementation. The HLLC flux collaborates with `EulerFlux`, which implements the analytical Euler flux function based on a `UState`, used by the HLLC flux, cf. (van der Vegt and van der Ven, 2002b). On the other hand, `EulerFlux` uses an implementation of `NumericalEulerFlux` when computing the flux on an internal face. `EulerFlux` is, however, also abstract in its current form as its template method aspect still requires it to be equipped with functions that can extract a state (of the type specified by the template argument `VariableSet`) from the user's data.[9] While the time flux of the Euler equations is uniquely defined by causality and can thus be implemented by `EulerFlux`, taking care of spatial boundary conditions cannot. Different types of con-

[8]For brevity, in this subsection the template arguments are omitted from names that would require the specification of the arguments to denote a type.

[9]The current flux class design effectively relies on the states being supplied from the user's data; this turns out as a restriction for some purposes.


d: DimType
S: class
F: class
NumericalFlux
+operator()(vL: S, vR: S, n: Vec<d>, flux: F): void
d: DimType
S: class
NumericalEulerFlux
+operator()(vL: S, vR: S, n: Vec<d>, flux: FV): void
d: DimType
VariableSet: class
EulerHLLC
_idealGas: IdealGas*
+operator()(vL: VState<d>, vR: VState<d>, n: Vec<d>, flux: FV): void
chain of responsibility aspect,
allows any number of boundary conditions
d: DimType
S: class
F: class
PrescribedFluxBoundaryCondition
-_next: PrescribedFluxBoundaryCondition<d, S, F>*
-_bcFlag: BoundaryFlagType
+operator()(f: Face<d>, n: Vec<d>, p: RSP<d-1>, vL: S, flux: F): void
#computeBoundaryFlux(f: Face<d>, n: Vec<d>, p: RSP<d-1>, vL: S, flux: F)
template method aspect
d: DimType
VariableSet: class
EulerFlux
-_fluxBC: PrescribedFluxBoundaryCondition<d, VState<d>, F>*
-_flagPast: BCFlagType
-_flagFuture: BCFlagType
-_numFlux: NumericalEulerFlux<d>*
-_fluidPtr: FluidThermodynamics*
+operator()(f: Face<d>, n: Vec<d>, p: RSP<d-1>, flux: FV): void
+spaceEulerFluxTimesVector(p: NumType, U: UState<d>, n1: Vec<d-1>, flux: FV): void
#getStates(f: Face<d>, p: RSP<d-1>, vL: S, vR: S)
#getStatePresentTL(f: Face<d>, p: RSP<d-1>, vL: S)
#getStatePreviousTL(f: Face<d>, p: RSP<d-1>, vL: S)
template method aspect

```
if (f.bcFlag() == _flag)
 this->computeBoundaryFlux(f, n, p, vL, flux);
else
 if (_next) _next->operator()(f, nv, p, vL, flux);
 else throw UnhandledBC(f.bcFlag());
```

Figure 6.7: Implementation perspective UML diagram of flux-related classes for the Euler equations. The following abbreviations are used: `Vec<d>`: abbreviation for `Base::FixedVector<NumType, d>`, an array of length d, `FV`: flux result type declared in the class (array of length $d+1$), `VState<d>`: state type implied by `VariableSet`, `RSP<d-1>`: abbreviation for `Geometry::RefSpacePoint<d-1>`, i.e., a point in reference face coordinates. See the discussion on pages 129–131.

ditions have been discussed in Chapter 5 and an arbitrary number of them may be used for a test case. Therefore, `EulerFlux` links to objects of a type that implements the `PrescribedFluxBoundaryCondition` interface. These objects treat different types of boundary conditions possible for the system, and are joined in a chain of responsibility, cf. (Gamma et al., 1994).

The described relationships and partitioning of responsibilities allow to vary every aspect of the flux computation independently. For the viscous terms of the Navier–Stokes case, boundary conditions are required as states rather than fluxes. For that purpose, a similar framework to the one presented here exists.

6.6 Conclusions

In this chapter, hpGEM, a general-purpose framework for the implementation of discontinuous Galerkin finite element discretizations, has been introduced. In contrast to several other finite element packages available, hpGEM is neither supposed to be a "solver" for a predefined set of equations nor is its focus to provide implementations of finite element algorithms. Rather it makes data structures and methods available that are general and frequently needed in the development and implementation of finite element application software. The framework provides means to define basis functions and expansions in terms of such bases, but leaves decisions about which data and variables to store to the user. Consequently, hpGEM does not impose restrictions on the type or number of equations solved. hpGEM works on general, unstructured meshes and all geometric transformations are handled internally. A convenient interface for the computation of integrals is provided.

Object-oriented programming techniques and design patterns are employed and lead to flexible, reusable, and easily extensible software. C++ templates are used to support generic programming aspects, like dimension-independent programming, user-defined data classes, and function return types. Their exploitation also allows to increase performance by template meta-programming techniques.

The framework does not implement any linear algebra solver capabilities but rather makes existing packages available, which provide well-tested, efficient iterative and direct solvers. The same holds for the necessary up- and downstream software, e.g., mesh generation and visualization tools.

The ongoing and future developments will make hpGEM a more versatile tool. The geometric capabilities of hpGEM will be extended by mesh refinement techniques to be used with general meshes. Apart from the more accurate geometric representation this also enables to apply multigrid techniques, which will increase the efficiency of the solution process. Expansion on the linear algebra back-ends is envisaged, too, so that a larger variety of sparse linear solvers—including parallel versions—are available to the framework user. These will allow to exploit parallelism in the solution stage, in addition to the parallelism in the finite element computation, which is another goal.

The hpGEM framework is an answer to the third central question from Chapter 1: it emphasizes combining self-developed FE-related components with existing solutions for supporting tasks and thereby provides numerical scientists with data structures and methods matching those from the mathematical formulation. A variety of PDE problems have been solved successfully and in concise form based on hpGEM, see also (Pesch et al., 2007), which underlines its suitability for numerical method development and application.

Chapter 7

Conclusions and recommendations

Der Weisheit erster Schritt ist: alles anzuklagen,
der letzte: sich mit allem zu vertragen.

Georg Christoph Lichtenberg (1742–1799),
Sudelbücher

At this stage it is time to summarize what has been achieved regarding the three central questions posed in Chapter 1 (pp. 3–4) of this thesis.

The starting point for the research presented here has been the realization that many numerical methods for the Navier–Stokes equations allow the treatment of only one specific type of fluid. Such a restriction is inappropriate when flow problems involving different fluids or physical conditions are to be addressed. Motivated by this shortcoming, in Chapter 2 the idea to consider the Navier–Stokes equations in terms of generalized variable sets—and in particular entropy variables—has been taken up. For entropy (and pressure primitive) variables the incompressible limit of the equations is well-defined, which allows suitably constructed numerical methods to compute both high and low Mach number flows, and even incompressible problems can be considered. As such, the formulation in terms of entropy variables provides an answer to the first central question.

In practical respects, an open challenge is to see how the formulation performs in combination with other fluid models than the ones employed here. For example, in the case of water a very small but nonzero compressibility applies. In general, it would be interesting to simulate fluids for which the effects not represented in the 'ideal' models of Chapter 2 play a role. In this respect, also the generalized equation of state as, e.g., presented by Polner (2005) could be a starting point. All these ideas, however, share two complications: First, the used fluid models have to be known in sufficient mathematical detail to allow all transformations necessary for the usage of entropy or pressure primitive variables. Second, all transformation functions have to be consistent with each other. This is expected to be hard to realize for measured equations of state, based on data tables.

The Galerkin least-squares discretization described in Chapter 4 is a numerical method that allows computations with the entropy variable formulation of the Navier–Stokes equations. It provides a well-established testbed for the application to compressible and incompressible flow. This setup has been extended by devising a stabilization operator that is applicable to both types of flow, see (Polner et al., 2007). The operator has been tested in various applications. It has been demonstrated that it does not degrade the accuracy of the method, which again holds for compressible and incompressible flow. Although this is an important step towards a general simulation tool, the second central question caused the investigations to turn into a different direction: The flexibility of discontinuous Galerkin methods with respect to local mesh refinement and (locally) higher order expansions meets the formulated requirements much better. Further, the unified space-time treatment and the solution with pseudo-time integration fit well into the target specification.

Having extended a discontinuous Galerkin method for ideal gas compressible flow to use general variable sets and different thermodynamical descriptions, the pseudo-time integration had to be adapted and extended in Chapter 5. With the discussed changes it allows to solve the nonlinear algebraic system arising from the discretization with various sets of independent variables. The class of problems the method is able to address has been extended by augmenting the initially used Euler model with the viscous terms of the Navier–Stokes equations. These have been treated using the interior penalty method. A wide range of test cases has been addressed and confirmed the accuracy of the numerical method as well as its applicability to very different physical settings. The discussed discontinuous Galerkin method has all the properties demanded by the second central question. However, its inspection has also raised several questions and left room for improvements. Regarding the pseudo-time integration, an advantage is that it does not require a linearization of the discrete operator. With more complex physical situations this will admittedly be an even more beneficial trait. On the other hand, the examination of the eigenvalue spectrum of the Euler operator and the experience from the simulations leaves some optimization to be desired. Convergence can be rather slow, and the analysis concerning the incompressible case has shown a particular need for improvements. Investigations in a preconditioner that adapts the spectrum better to the available Runge–Kutta methods might lead to better performance. It should be kept in mind, however, that the solution method must not restrict the applicability to specific media. Ideally the employed techniques should be robust enough to work with compressible and incompressible fluids. In general the discussed finite element method could benefit greatly from applying an h-multigrid algorithm as developed by Klaij et al. (2007).

In combination with an arbitrary Lagrangian–Eulerian formulation, the space-time method naturally accommodates moving and deforming meshes, which is a good starting point for dynamic adaptation and interface tracking. For the latter task, the treatment in space-time, unfortunately, increases the complexity considerably. Refining the elements in such a way that the fluid-fluid interface is a face is a nontrivial task in two space dimensions. For three-dimensional space meshes the problem is yet unsolved: the added time dimension means that four-dimensional geometries have to be partitioned based on the intersecting interface.

The numerical methods discussed in this dissertation lead to complex solution algorithms that have to be implemented as software to be applied in practice. The third central question sought for a facilitation of the implementation task and has led to the work on hpGEM, a general-purpose discontinuous Galerkin finite element method software framework. Chapter 6 has documented the ideas behind hpGEM and cornerstones of its design and implementation. Useful paradigms and methodologies have been presented and their benefit explained. The confirmation that hpGEM's goals are achieved is included in Chapter 5: the implementation of the numerical method presented there is built on hpGEM. Beyond that, by now about half a dozen PhD students and researchers have implemented different finite element methods for various mathematical problems based on the framework.

Although this gives a positive feedback for the development, a lot remains to be done. In Chapter 6, several concept areas have been pointed out that require extensions. A major step necessary for several key technologies is to enable dynamic local mesh refinement. It would not only facilitate multigrid techniques but also the adaptation to fine scale features and (space-time) interface tracking, which are among the visions for the overall project. To realize this feature with minimal impact on existing applications based on hpGEM and to provide an easy-to-use interface should be one of the next challenges. The discontinuous Galerkin method for the Navier–Stokes equations has raised the need for performance improvements. Several steps in this direction have already been taken, but ultimately the limit will prove the sequential processing to which framework is so far restricted. The longer this circumstance lasts, the more it aggravates, as the ongoing developments rarely facilitate or even consider the problems inherent in parallelization. A more fundamental problem for the framework development is the supply of expertise: for the continuation of the project, contributors with an understanding of both the mathematical topics and object-oriented design and programming are required.

Having formulated answers to the three central questions and delivered a proof of concept for the developed numerical method and software artefacts, the result of this thesis is a promising path for future developments of finite element methods for multiphase flow simulation.

Appendix A

Matrices for the quasi-linearized form of the Navier–Stokes equations

In this appendix, the derivation of the matrices $A^V_{\bar{\imath}}$ and $K^V_{\bar{\imath}\bar{\jmath}}$ of the quasi-linear form of the Navier–Stokes equations (cf. Section 2.8.1) is summarized for pressure primitive and entropy variables. The derivations are carried out using the dimensional rather than the nondimensional form of the equations, cf. Section 2.10. Only general thermodynamical relations from Section 2.6 are used, without assuming a specific equation of state.

A.1 Primitive variables using pressure

A.1.1 Jacobian A^Y_0 of the transformation $U(Y)$

The partial derivatives of the conservation variables U with respect to primitive variables Y (cf. (2.44,2.46) on page 28) are evaluated using (2.25,2.27,2.29).

$$\left(\frac{\partial\rho}{\partial p}\right)_{T,v_{\bar{\imath}}} = \left(\frac{\partial\alpha^{-1}}{\partial p}\right)_{T,v_{\bar{\imath}}} = -\alpha^{-2}\left(\frac{\partial\alpha}{\partial p}\right)_{T,v_{\bar{\imath}}} = \frac{\beta_T}{\alpha} = \rho\beta_T\,, \tag{A.1a}$$

$$\left(\frac{\partial(\rho v_{\bar{\imath}})}{\partial p}\right)_{T,v_{\bar{\imath}}} = v_{\bar{\imath}}\left(\frac{\partial\rho}{\partial p}\right)_{T,v_{\bar{\imath}}} = \frac{\beta_T v_{\bar{\imath}}}{\alpha} = \rho\beta_T v_{\bar{\imath}}\,, \tag{A.1b}$$

$$\left(\frac{\partial(\rho e^{\text{tot}})}{\partial p}\right)_{T,v_{\bar{\imath}}} = \frac{\beta_T}{\alpha}e^{\text{tot}} + \rho\left(\frac{\partial e}{\partial p}\right)_{T,v_{\bar{\imath}}} = \beta_T(\rho e^{\text{tot}} + p) - T\alpha_p\,, \tag{A.1c}$$

$$\left(\frac{\partial\rho}{\partial v_{\bar{\jmath}}}\right)_{p,T} = 0\,, \tag{A.1d}$$

$$\left(\frac{\partial(\rho v_{\bar{\imath}})}{\partial v_{\bar{\jmath}}}\right)_{p,T} = \rho\delta_{\bar{\imath}\bar{\jmath}}\,, \tag{A.1e}$$

$$\left(\frac{\partial(\rho e^{\text{tot}})}{\partial v_{\bar{\jmath}}}\right)_{p,T} = \rho v_{\bar{\jmath}}\,, \tag{A.1f}$$

$$\left(\frac{\partial\rho}{\partial T}\right)_{p,v_{\bar{j}}} = \left(\frac{\partial\alpha^{-1}}{\partial T}\right)_{p,v_{\bar{\imath}}} = -\alpha^{-2}\left(\frac{\partial\alpha}{\partial T}\right)_{p,v_{\bar{\imath}}} = -\frac{\alpha_p}{\alpha} = -\rho\alpha_p\,, \tag{A.1g}$$

$$\left(\frac{\partial(\rho v_{\bar{\imath}})}{\partial T}\right)_{p,v_{\bar{j}}} = \left(\frac{\partial\rho}{\partial T}\right)_{p,v_{\bar{j}}} v_{\bar{\imath}} = -\frac{\alpha_p v_{\bar{\imath}}}{\alpha} = -\rho\alpha_p v_{\bar{\imath}}\,, \tag{A.1h}$$

$$\left(\frac{\partial(\rho e^{\mathrm{tot}})}{\partial T}\right)_{p,v_{\bar{j}}} = -\alpha_p(\rho e^{\mathrm{tot}} + p) + \rho c_{\mathrm{p}}\,. \tag{A.1i}$$

Therefore the Jacobian matrix is

$$A_0^Y = \frac{\partial U}{\partial Y} = \begin{pmatrix} \rho\beta_T & 0 & -\rho\alpha_p \\ \rho\beta_T v_{\bar{\imath}} & \rho\delta_{\bar{\imath}\bar{j}} & -\rho\alpha_p v_{\bar{\imath}} \\ \beta_T(\rho e^{\mathrm{tot}} + p) - T\alpha_p & \rho v_{\bar{j}} & -\alpha_p(\rho e^{\mathrm{tot}} + p) + \rho c_{\mathrm{p}} \end{pmatrix}. \tag{A.2}$$

A.1.2 Euler flux Jacobians $A_{\bar{k}}^Y = \partial F_{\bar{k}}^{\mathrm{e}}/\partial Y$

The Euler flux in the $\bar{k}^{\mathrm{th}}$ Cartesian coordinate direction is repeated from Eq. (2.15):

$$F_{\bar{k}}^{\mathrm{e}} = \begin{pmatrix} \rho v_{\bar{k}} \\ \rho v_{\bar{\imath}} v_{\bar{k}} + p\delta_{\bar{\imath}\bar{k}} \\ (\rho e^{\mathrm{tot}} + p) v_{\bar{k}} \end{pmatrix}. \tag{A.3}$$

The partial derivatives of the flux components with respect to the primitive variables are

$$\left(\frac{\partial(\rho v_{\bar{k}})}{\partial p}\right)_{T,v_{\bar{j}}} = \rho\beta_T v_{\bar{k}}\,, \tag{A.4a}$$

$$\left(\frac{\partial(\rho v_{\bar{\imath}} v_{\bar{k}} + p\delta_{\bar{\imath}\bar{k}})}{\partial p}\right)_{T,v_{\bar{j}}} = \rho\beta_T v_{\bar{\imath}} v_{\bar{k}} + \delta_{\bar{\imath}\bar{k}}\,, \tag{A.4b}$$

$$\left(\frac{\partial((\rho e^{\mathrm{tot}} + p)v_{\bar{k}})}{\partial p}\right)_{T,v_{\bar{j}}} = [\beta_T(\rho e^{\mathrm{tot}} + p) - \alpha_p T + 1]v_{\bar{k}}\,, \tag{A.4c}$$

$$\left(\frac{\partial(\rho v_{\bar{k}})}{\partial v_{\bar{j}}}\right)_{p,T} = \rho\delta_{\bar{k}\bar{j}}\,, \tag{A.4d}$$

$$\left(\frac{\partial(\rho v_{\bar{\imath}} v_{\bar{k}} + p\delta_{\bar{\imath}\bar{k}})}{\partial v_{\bar{j}}}\right)_{p,T} = \rho(\delta_{\bar{\imath}\bar{j}} v_{\bar{k}} + \delta_{\bar{k}\bar{j}} v_{\bar{\imath}})\,, \tag{A.4e}$$

$$\left(\frac{\partial((\rho e^{\mathrm{tot}} + p)v_{\bar{k}})}{\partial v_{\bar{j}}}\right)_{p,T} = \rho v_{\bar{j}} v_{\bar{k}} + (\rho e^{\mathrm{tot}} + p)\delta_{\bar{k}\bar{j}}\,, \tag{A.4f}$$

$$\left(\frac{\partial\rho v_{\bar{k}}}{\partial T}\right)_{p,v_{\bar{j}}} = -\rho\alpha_p v_{\bar{k}}\,, \tag{A.4g}$$

$$\left(\frac{\partial((\rho v_{\bar{\imath}} v_{\bar{k}} + p\delta_{\bar{\imath}\bar{k}})}{\partial T}\right)_{p,v_{\bar{\jmath}}} = -\rho\alpha_p v_{\bar{\imath}} v_{\bar{k}}\,, \tag{A.4h}$$

$$\left(\frac{\partial(\rho e^{\mathrm{tot}} + p)v_{\bar{k}})}{\partial T}\right)_{p,v_{\bar{\jmath}}} = [-\alpha_p(\rho e^{\mathrm{tot}} + p) + \rho c_{\mathrm{p}}]v_{\bar{k}}\,. \tag{A.4i}$$

With these findings, the Jacobian is compiled as

$$A^Y_{\bar{k}} = \frac{\partial F^{\mathrm{e}}_{\bar{k}}}{\partial Y} = \tag{A.5}$$
$$\begin{pmatrix} \rho\beta_T v_{\bar{k}} & \rho\delta_{\bar{k}\bar{\jmath}} & -\rho\alpha_p v_{\bar{k}} \\ \rho\beta_T v_{\bar{\imath}} v_{\bar{k}} + \delta_{\bar{\imath}\bar{k}} & \rho(\delta_{\bar{\imath}\bar{\jmath}} v_{\bar{k}} + \delta_{\bar{k}\bar{\jmath}} v_{\bar{\imath}}) & -\rho\alpha_p v_{\bar{\imath}} v_{\bar{k}} \\ [\beta_T(\rho e^{\mathrm{tot}} + p) - \alpha_p T + 1]v_{\bar{k}} & \rho v_{\bar{\jmath}} v_{\bar{k}} + (\rho e^{\mathrm{tot}} + p)\delta_{\bar{k}\bar{\jmath}} & [-\alpha_p(\rho e^{\mathrm{tot}} + p) + \rho c_{\mathrm{p}}]v_{\bar{k}} \end{pmatrix}.$$

A.1.3 Viscous flux Jacobians $K^Y_{\bar{\imath}\bar{\jmath}}$

For a single, homogeneous fluid there is no mass diffusion, which is reflected in the fact that the diffusive fluxes in the continuity equation vanish. Hence the first row in the diffusivity matrices $K^Y_{\bar{\imath}\bar{\jmath}}$ is zero.

Considering, for the moment, only the momentum equations, the viscous terms take the form discussed in Section 2.2,

$$\nabla \cdot \mathbb{P}_{\mathrm{visc}} = \frac{\partial F^{\mathrm{d}}_{\bar{n}}}{\partial x_{\bar{n}}} = \frac{\partial}{\partial x_{\bar{m}}}\left(K^Y_{\bar{m}\bar{n}}\frac{\partial Y}{\partial x_{\bar{n}}}\right), \tag{A.6}$$

with the diffusive flux of the $\bar{\imath}^{\mathrm{th}}$ momentum component in the $\bar{\jmath}^{\mathrm{th}}$ Cartesian coordinate direction given by Eq. (2.14b),

$$F^{\mathrm{d}}_{\bar{\imath}\bar{\jmath}} = (\lambda - \tfrac{2}{3}\eta)\frac{\partial v_{\bar{n}}}{\partial x_{\bar{n}}}\delta_{\bar{\imath}\bar{\jmath}} + \eta\left(\frac{\partial v_{\bar{\jmath}}}{\partial x_{\bar{\imath}}} + \frac{\partial v_{\bar{\imath}}}{\partial x_{\bar{\jmath}}}\right). \tag{A.7}$$

Taking F^{d}_1 in dimension $d = 3$ as an example, the terms of the flux

$$F^{\mathrm{d}}_1 = (\lambda - \tfrac{2}{3}\eta)\frac{\partial v_n}{\partial x_n}\begin{pmatrix}1\\0\\0\end{pmatrix} + \eta\begin{pmatrix}\frac{\partial v_1}{\partial x_1} + \frac{\partial v_1}{\partial x_1}\\ \frac{\partial v_2}{\partial x_1} + \frac{\partial v_1}{\partial x_2}\\ \frac{\partial v_3}{\partial x_1} + \frac{\partial v_1}{\partial x_3}\end{pmatrix}, \tag{A.8}$$

can be assigned to the linearization matrices according to

$$K^Y_{11} = \begin{pmatrix}\lambda - \frac{2}{3}\eta + 2\eta & 0 & 0\\ 0 & \eta & 0\\ 0 & 0 & \eta\end{pmatrix},\; K^Y_{12} = \begin{pmatrix}0 & \lambda - \frac{2}{3}\eta & 0\\ \eta & 0 & 0\\ 0 & 0 & 0\end{pmatrix},\; K^Y_{13} = \begin{pmatrix}0 & 0 & \lambda - \frac{2}{3}\eta\\ 0 & 0 & 0\\ \eta & 0 & 0\end{pmatrix}. \tag{A.9}$$

In general, by comparing the components of $F^{\mathsf{d}}_{\bar{k}\bar{\imath}} = K^Y_{\bar{\imath}\bar{\jmath}\bar{k}\bar{l}} Y_{\bar{l},\bar{\jmath}}$, one finds the entry of the $\bar{k}^{\text{th}}$ row and $\bar{l}^{\text{th}}$ column of the $d \times d$ momentum block of $K^Y_{\bar{\imath}\bar{\jmath}}$ to be

$$K^Y_{\bar{\imath}\bar{\jmath}\bar{k}\bar{l}} = \delta_{\bar{\imath}\bar{k}}\delta_{\bar{\jmath}\bar{l}}(\lambda - \tfrac{2}{3}\eta) + \eta(\delta_{\bar{k}\bar{l}}\delta_{\bar{\imath}\bar{\jmath}} + \delta_{\bar{\imath}\bar{l}}\delta_{\bar{\jmath}\bar{k}})\,. \tag{A.10}$$

The first and last column of $K^Y_{\bar{\imath}\bar{\jmath}}$ are zero in the momentum rows because the viscous momentum fluxes neither depend on pressure nor temperature.

Further, for the energy equation, the non-advective contributions are dissipation and heat conduction:

$$\nabla \cdot (\mathbb{P}_{\mathsf{visc}} \cdot \boldsymbol{v}) - \nabla \cdot \boldsymbol{q} = \nabla \cdot \left[(\lambda - \tfrac{2}{3}\eta)(\nabla \cdot \boldsymbol{v})\boldsymbol{v} + \eta(\nabla\boldsymbol{v} + (\nabla\boldsymbol{v})^{\mathsf{T}}) \cdot \boldsymbol{v} + \kappa\nabla T\right]. \tag{A.11}$$

Hence, the flux in the $\bar{\imath}^{\text{th}}$ Cartesian coordinate is

$$F^{\mathsf{d}}_{(d+2)\bar{\imath}} = (\lambda - \tfrac{2}{3}\eta)\frac{\partial v_{\bar{n}}}{\partial x_{\bar{n}}} v_{\bar{\imath}} + \eta\left(\frac{\partial v_{\bar{n}}}{\partial x_{\bar{\imath}}} + \frac{\partial v_{\bar{\imath}}}{\partial x_{\bar{n}}}\right) v_{\bar{n}} + \kappa\frac{\partial T}{\partial x_{\bar{\imath}}}\,, \tag{A.12}$$

which reappears in the last row of the $K^Y_{\bar{\imath}\bar{\jmath}}$ matrices as[1]

$$K^Y_{\bar{\imath}\bar{\jmath}(d+2)\tilde{l}} = \begin{cases} 0 & \text{if } \tilde{l} = 0\,, \\ (\lambda - \tfrac{2}{3}\eta)\delta_{\bar{\jmath}\tilde{l}} v_{\bar{\imath}} + \eta(\delta_{\bar{\imath}\bar{\jmath}} v_{\tilde{l}} + \delta_{\bar{\imath}\tilde{l}} v_{\bar{\jmath}}) & \text{if } 0 < \tilde{l} < 1 + d\,, \\ \kappa\delta_{\bar{\imath}\bar{\jmath}} & \text{if } \tilde{l} = 1 + d\,. \end{cases} \tag{A.13}$$

For the spatial dimension $d = 3$ the matrices $K^Y_{\bar{\imath}\bar{\jmath}}$ are written out in Table A.1 on the facing page.

A.2 Entropy variables

Because many differential thermodynamical relations are in terms of the (observable) variables pressure and temperature, transforming from entropy to primitive variables first facilitates the step to introduce entropy variables. For example, the way to introduce derivatives of the entropy variables in the time derivative is

$$\frac{\partial U}{\partial t} = \frac{\partial U}{\partial Y}\frac{\partial Y}{\partial V}\frac{\partial V}{\partial t} = \frac{\partial U}{\partial Y}\left(\frac{\partial V}{\partial Y}\right)^{-1}\frac{\partial V}{\partial t} = A^V_0\frac{\partial V}{\partial t}\,. \tag{A.14}$$

[1] The variable index $\tilde{l}$ is assumed to cover the range $0, \ldots, 1 + d$ here; in this case the space dimension numbering can remain as $\bar{\imath}, \bar{\jmath} = 1, \ldots, d$.

$\bar{\jmath}\rightarrow$ $\downarrow\bar{\imath}$	1	2	3
1	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \chi & 0 & 0 & 0 \\ 0 & 0 & \eta & 0 & 0 \\ 0 & 0 & 0 & \eta & 0 \\ 0 & \chi v_1 & \eta v_2 & \eta v_3 & \kappa \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \theta & 0 & 0 \\ 0 & \eta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \eta v_2 & \theta v_1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \theta & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \eta & 0 & 0 & 0 \\ 0 & \eta v_3 & 0 & \theta v_1 & 0 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \eta & 0 & 0 \\ 0 & \theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \theta v_2 & \eta v_1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \eta & 0 & 0 & 0 \\ 0 & 0 & \chi & 0 & 0 \\ 0 & 0 & 0 & \eta & 0 \\ 0 & \eta v_1 & \chi v_2 & \eta v_3 & \kappa \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \theta & 0 \\ 0 & 0 & \eta & 0 & 0 \\ 0 & 0 & \eta v_3 & \theta v_2 & 0 \end{bmatrix}$
3	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \eta & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \theta & 0 & 0 & 0 \\ 0 & \theta v_3 & 0 & \eta v_1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \eta & 0 \\ 0 & 0 & \theta & 0 & 0 \\ 0 & 0 & \theta v_3 & \eta v_2 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \eta & 0 & 0 & 0 \\ 0 & 0 & \eta & 0 & 0 \\ 0 & 0 & 0 & \chi & 0 \\ 0 & \eta v_1 & \eta v_2 & \chi v_3 & \kappa \end{bmatrix}$

Table A.1: Viscous flux matrices $K^Y_{\bar{\imath}\bar{\jmath}}$ for primitive variables with pressure and space dimension $d = 3$. The following coefficients are defined based on the viscosity coefficients: $\chi = \lambda + 4/3\,\eta$ and $\theta = \lambda - 2/3\,\eta$.

A.2.1 Jacobian A_0^V of the transformation $U(V)$

In Equation (A.14), the Jacobian $\partial V/\partial Y$ is yet unknown. The necessary derivatives are

$$\left(\frac{\partial((\mu - v_n^2/2)/T)}{\partial p}\right)_{T,v_{\bar{\imath}}} = \frac{1}{T}\left(\frac{\partial \mu}{\partial p}\right)_{T,v_{\bar{\imath}}} = \frac{\alpha}{T}\,, \tag{A.15a}$$

$$\left(\frac{\partial(v_{\bar{\imath}}/T)}{\partial p}\right)_{T,v_{\bar{\jmath}}} = 0\,, \tag{A.15b}$$

$$\left(\frac{\partial(-1/T)}{\partial p}\right)_{T,v_{\bar{\imath}}} = 0\,, \tag{A.15c}$$

$$\left(\frac{\partial((\mu - v_n^2/2)/T)}{\partial v_{\bar{\jmath}}}\right)_{p,T} = -\frac{v_{\bar{\jmath}}}{T}\,, \tag{A.15d}$$

$$\left(\frac{\partial(v_{\bar{\imath}}/T)}{\partial v_{\bar{\jmath}}}\right)_{p,T} = \frac{1}{T}\delta_{\bar{\imath}\bar{\jmath}}\,, \tag{A.15e}$$

$$\left(\frac{\partial(-1/T)}{\partial v_{\bar{\imath}}}\right)_{p,T} = 0\,, \tag{A.15f}$$

$$\left(\frac{\partial((\mu - v_n^2/2)/T)}{\partial T}\right)_{p,v_{\bar{\imath}}} = \frac{1}{T}\left(\frac{\partial \mu}{\partial T}\right)_{p,v_{\bar{\imath}}} - \frac{1}{T^2}(\mu - v_n^2/2) = -\frac{1}{T^2}(h-k)\,, \tag{A.15g}$$

$$\left(\frac{\partial(v_{\bar{\imath}}/T)}{\partial T}\right)_{p,v_{\bar{\jmath}}} = -\frac{v_{\bar{\imath}}}{T^2}\,, \tag{A.15h}$$

$$\left(\frac{\partial(-1/T)}{\partial T}\right)_{p,v_{\bar{\imath}}} = \frac{1}{T^2}\,. \tag{A.15i}$$

Therefore, the Jacobian of the transformation $V(Y)$ is

$$\frac{\partial V}{\partial Y} = \frac{1}{T}\begin{pmatrix} \alpha & -v_{\bar{\jmath}} & -(h-k)/T \\ 0 & \delta_{\bar{\imath}\bar{\jmath}} & -v_{\bar{\imath}}/T \\ 0 & 0 & 1/T \end{pmatrix}, \tag{A.16}$$

with the inverse

$$\frac{\partial Y}{\partial V} = T\begin{pmatrix} \rho & \rho v_{\bar{\jmath}} & \rho(h+k) \\ 0 & \delta_{\bar{\imath}\bar{\jmath}} & v_{\bar{\imath}} \\ 0 & 0 & T \end{pmatrix}. \tag{A.17}$$

Concatenating with the transformation to conservation variables, (A.2), the relevant Jacobian is obtained following (A.14) as

$$A_0^V = \rho^2 T\begin{pmatrix} \beta_T & \beta_T v_{\bar{\jmath}} & \beta_T(h+k) - \alpha\alpha_p T \\ & \beta_T v_{\bar{\imath}} v_{\bar{\jmath}} + \alpha\delta_{\bar{\imath}\bar{\jmath}} & [\beta_T(h+k) - \alpha(\alpha_p T - 1)]v_{\bar{\imath}} \\ \text{sym.} & & (h+k)[\beta_T(h+k) - 2\alpha\alpha_p T] + \alpha(c_\mathrm{p} T + 2k) \end{pmatrix}. \tag{A.18}$$

A.2.2 Euler flux Jacobians $A_{\bar{k}}^V = \partial F_{\bar{k}}^\mathrm{e}/\partial V$

Analogously to A_0^V, the Euler flux Jacobian $A_{\bar{k}}^V$ for entropy variables is found by postmultiplying (A.5) with (A.17), which results in the matrix in (A.19) on page 144.

A.2.3 Viscous flux Jacobians $K_{\bar{\imath}\bar{\jmath}}^V$

Previously, the viscosity matrices $K_{\bar{\imath}\bar{\jmath}}^Y$ for primitive variables using pressure have been derived; the matrices $K_{\bar{\imath}\bar{\jmath}}^V$ for entropy variables can be obtained by postmultiplication with the inverse Jacobian of the variable transformation, cf. (A.17). These matrices are listed in Table A.2 on the following page for the spatial dimension $d = 3$.

$\bar{j}\rightarrow$ $\downarrow\bar{\imath}$	1	2	3
1	$T\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \chi & 0 & 0 & \chi v_1 \\ 0 & 0 & \eta & 0 & \eta v_2 \\ 0 & 0 & 0 & \eta & \eta v_3 \\ 0 & \chi v_1 & \eta v_2 & \eta v_3 & \omega {v_1}^2 + 2\eta k + \kappa T \end{bmatrix}$	$T\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \theta & 0 & \theta v_2 \\ 0 & \eta & 0 & 0 & \eta v_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \eta v_2 & \theta v_1 & 0 & \omega v_1 v_2 \end{bmatrix}$	$T\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \theta & \theta v_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \eta & 0 & 0 & \eta v_1 \\ 0 & \eta v_3 & 0 & \theta v_1 & \omega v_1 v_3 \end{bmatrix}$
2	$K^V_{21} = (K^V_{12})^\intercal$	$T\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \eta & 0 & 0 & \eta v_1 \\ 0 & 0 & \chi & 0 & \chi v_2 \\ 0 & 0 & 0 & \eta & \eta v_3 \\ 0 & \eta v_1 & \chi v_2 & \eta v_3 & \omega {v_2}^2 + 2\eta k + \kappa T \end{bmatrix}$	$T\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \theta & \theta v_3 \\ 0 & 0 & \eta & 0 & \eta v_2 \\ 0 & 0 & \eta v_3 & \theta v_2 & \omega v_2 v_3 \end{bmatrix}$
3	$K^V_{31} = (K^V_{13})^\intercal$	$K^V_{32} = (K^V_{23})^\intercal$	$T\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \eta & 0 & 0 & \eta v_1 \\ 0 & 0 & \eta & 0 & \eta v_2 \\ 0 & 0 & 0 & \chi & \chi v_3 \\ 0 & \eta v_1 & \eta v_2 & \chi v_3 & \omega {v_3}^2 + 2\eta k + \kappa T \end{bmatrix}$

Table A.2: Viscous flux matrices $K^V_{\bar{\imath}\bar{j}}$ for entropy variables and space dimension $d = 3$. The following coefficients are defined based on the viscosity coefficients: $\chi = \lambda + 4/3\,\eta$, $\theta = \lambda - 2/3\,\eta$, and $\omega = \lambda + 1/3\,\eta$.

$$A^V_{\bar{k}} = \rho^2 T \begin{pmatrix} \beta_T v_{\bar{k}} & \beta_T v_{\bar{k}} v_{\bar{j}} + \alpha\delta_{\bar{j}\bar{k}} & v_{\bar{k}}[\beta_T(h+k) - \alpha(\alpha_p T - 1)] \\ & v_{\bar{k}}(\beta_T v_{\bar{\imath}} v_{\bar{j}} + \alpha(\delta_{\bar{\imath}\bar{j}+} + \delta_{\bar{\imath}\bar{k}} + \delta_{\bar{j}\bar{k}}) & v_{\bar{k}}[\beta_T(h+k) - \alpha(\alpha_p T - 2)]v_{\bar{\imath}} + \delta_{\bar{\imath}\bar{k}}\alpha(h+k) \\ \text{sym.} & & v_{\bar{k}}[\beta_T(h+k)^2 + \alpha(2(h+k)(1-\alpha_p T) + 2k + c_{\mathrm{p}}T)] \end{pmatrix}. \tag{A.19}$$

Appendix B
hpGEM geometry details

In Chapter 3, the element shapes supported by hpGEM in meshes of dimension one to three have been listed. Each reference geometry is defined in an implementation of the abstract base class template `ReferenceGeometry<dim>`.[1] In the following sections, these will be described in more detail, adding information about the mappings and integration rules provided by the framework.

Mappings $G\colon \hat{\mathcal{K}} \to \mathcal{K}$ from the reference geometry to physical space are constructed with the nodal basis for the vertices of each shape. The reference shape vertices are given by their Cartesian coordinates $\hat{\boldsymbol{x}} = (\hat{x}_0, \dots, \hat{x}_{d-1})$. For each reference space vertex $\hat{\boldsymbol{x}}_i$ the physical space point is $\boldsymbol{x}_i$. No distinction is made in the current chapter between space-time and space only meshes as these are merely different interpretations of a given tesselation.

Numerical integration rules are implemented based on (Stroud, 1971) and their naming follows the scheme introduced there. For each rule, the relevant page number in the cited work is given for quick reference.

The data related to faces (cf. p. 42) is a part of the geometry information provided by hpGEM. The faces of a reference shape are defined by an ordered subset of its vertices, where the order follows the ordering of vertices in the lower-dimensional reference geometry of the face (for example, topologically the faces of the reference triangle are reference lines). The face integrals in finite element formulations require the (solution) data to be evaluated on the faces, hence points on the face reference geometry have to be mapped to the (reference) geometry of the element. Providing these mappings is the most important task for the faces. Concatenated with the element mapping to physical space, the face mapping also incorporates the information about the face normal vector. Depending on the ordering of the face vertices, the normal vector computed from the Jacobian of the face to element mapping can be either inward or outward. To guarantee that the normal vectors delivered to user code are always outward, the geometry description includes a sign for the normal vector coordinates on each face.[2]

[1] The class template `ReferenceGeometry` in turn inherits an interface for the description of the composition in terms of lower-dimensional geometric entities from `CodimMaps<dim>` and an implementation component from `GaussRuleServer<dim>`, which enables the implementation classes to manage the numerical integration rules related to the reference geometry. A similar functionality should be added for different mappings.

[2] Alternatively the ordering of the face vertices could be changed. In cases where the face to element mappings originate from legacy code this has note been done.

Edges, i.e., connections between two nodes, are the natural entities of codimension two implied by a mesh. For three-dimensional reference geometries, these are defined as well. However so far no edge-to-element mappings are provided by hpGEM, so the definition of the edges is provisional.

B.1 One-dimensional reference geometry

B.1.1 `ReferenceLine`

One-dimensional meshes cover a line segment and a single reference geometry suffices for this purpose. `ReferenceLine` is implemented mainly for consistency and for verification purposes, since one-dimensional geometries are of small practical relevance and do not even exist as elements in the space-time context.

Geometry and topology

`ReferenceLine` represents the one-dimensional instance of the standard d-cube, which is defined as $\hat{\mathcal{K}} = [-1; 1]^d$ here. In this way, the coordinates of Gauß integration points normally exhibit symmetry with respect to the coordinate axes. The two vertices of the reference line at $\hat{\boldsymbol{x}}_0 = (-1)$ and $\hat{\boldsymbol{x}}_1 = (+1)$ can also be interpreted as faces in the notion of Chapter 3 and the normal vectors are $\boldsymbol{n}_0 = (-1)$ and $\boldsymbol{n}_1 = (+1)$.

Mapping

The standard mapping of the reference line is the linear interpolation between the two physical space points $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$:

$$G(\hat{x}) = \frac{1 - \hat{x}}{2}\boldsymbol{x}_0 + \frac{1 + \hat{x}}{2}\boldsymbol{x}_1 \,. \tag{B.1}$$

Integration rules

The integration rules that are implemented for the `ReferenceLine` are listed in Table B.1.

class name	order	# points	remark
`Cn1_1_1`	1	1	centroid formula, p. 229
`Cn1_3_4`	3	2	Gauß formula, p. 230
`Cn1_5_9`	5	3	Gauß formula, p. 234
`C1_7_x`	7	4	given without name on p. 314

Table B.1: Integration rules related to `ReferenceLine`.

B.2 Two-dimensional reference geometries

In two-dimensional space, the common element geometries are triangles and quadrilaterals. Their reference geometries are presented next.

B.2.1 `ReferenceTriangle`

`ReferenceTriangle` provides the two-dimensional version of the d-simplex, a shape that is defined by the origin and all points at unit distance from it on the positive coordinate axes.

Geometry and topology

The vertex positions of the reference triangle have been described above and are written out in Table B.2a. For two-dimensional shapes, faces of nonzero measure arise, which are described in Table B.2b.

i	$\hat{\boldsymbol{x}}_i$
0	$(0,0)$
1	$(1,0)$
2	$(0,1)$

(a)

face #	$(i_1,\dots)\colon \hat{\boldsymbol{x}}_i \in \mathcal{S}$		sign	remark
0	0	1	-1	$\hat{x}_1 = 0$
1	0	2	$+1$	$\hat{x}_0 = 0$
2	1	2	-1	$\hat{x}_0 + \hat{x}_1 = 1$

(b)

Table B.2: `ReferenceTriangle` definitions, (a) vertex coordinates, (b) faces.

Mapping

For the reference d-simplex $\hat{\mathcal{K}}$, a linear mapping to a simplex $\mathcal{K}$ with the vertices $\boldsymbol{x}_i$, $i = 0,\dots,d$, is given by

$$G(\hat{\boldsymbol{x}}) = \boldsymbol{x}_0 + \sum_{i=1}^{d} \hat{x}_i(\boldsymbol{x}_i - \boldsymbol{x}_0)\,. \tag{B.2}$$

Integration rules

The integration rules that are implemented for the `ReferenceTriangle` are listed in Table B.3 on the next page.

B.2.2 `ReferenceSquare`

The reference square can be interpreted as a Cartesian product of two reference lines, one on the $\hat{x}_0$-axis and one on the $\hat{x}_1$-axis. Thereby some of the definitions resemble the

class name	order	# points	remark
`Tn2_1_1`	1	1	midpoint formula, p. 307
`Tn2_2_1`	2	3	p. 307
`Tn2_3_1`	3	4	negative weight, p. 308
`T2_5_1`	5	7	p. 314
`T2_7_1`	7	16	conical product Gauß formula, p. 314, (see also the remarks about integration rules for d-dimensional cones in Section B.3.2)

Table B.3: Integration rules related to `ReferenceTriangle`.

one-dimensional case. In particular the integration rules are product Gauß quadrature rules exploiting this setup.

Geometry and topology

The vertex positions are clear based on the Cartesian product definition of `Reference-Square`. The enumeration of the nodes is per dimension starting in the one with the lowest Cartesian coordinate index, cf. Table B.4a. The four faces of the reference square are defined in Table B.4b.

i	$\hat{\boldsymbol{x}}_i$
0	$(-1,-1)$
1	$(+1,-1)$
2	$(-1,+1)$
3	$(+1,+1)$

(a)

face #	$(i_1,\ldots)$: $\hat{\boldsymbol{x}}_i \in \mathcal{S}$		sign	remark
0	0	1	-1	$\hat{x}_1 = -1$
1	0	2	$+1$	$\hat{x}_0 = -1$
2	1	3	-1	$\hat{x}_0 = +1$
3	2	3	$+1$	$\hat{x}_1 = +1$

(b)

Table B.4: `ReferenceSquare` definitions, (a) vertex coordinates, (b) faces.

Mapping

The standard mappings on the d-cube can be expressed by the nodal basis functions, i.e., as a weighted sum of the node vectors $\boldsymbol{x}_i$. Depending on the compromise between computational cost and memory requirement, rearranging the terms of the nodal mapping can be advantageous, as will be discussed here by example of the mapping for quadrilaterals. Based on the vertex positions $\boldsymbol{x}_i$, the following linear combinations are computed:

$$\boldsymbol{a} = \tfrac{1}{4}(+\boldsymbol{x}_0 + \boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3)\,, \quad \text{(B.3a)} \qquad \boldsymbol{a}_0 = \tfrac{1}{4}(-\boldsymbol{x}_0 + \boldsymbol{x}_1 - \boldsymbol{x}_2 + \boldsymbol{x}_3)\,, \quad \text{(B.3b)}$$

$$\boldsymbol{a}_1 = \tfrac{1}{4}(-\boldsymbol{x}_0 - \boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3)\,, \quad \text{(B.3c)} \qquad \boldsymbol{a}_{01} = \tfrac{1}{4}(+\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2 + \boldsymbol{x}_3)\,. \quad \text{(B.3d)}$$

With these auxiliary vectors, the mapping can be evaluated as

$$G(\hat{\boldsymbol{x}}) = \boldsymbol{a} + \hat{x}_0\, \boldsymbol{a}_0 + \hat{x}_1\, (\boldsymbol{a}_1 + \hat{x}_0\, \boldsymbol{a}_{01})\,. \tag{B.4}$$

At the cost of three multiplications of a vector with a scalar, three vector additions, and six temporaries (if expression (B.4) is evaluated as a binary expression tree—unlike the expression template techniques discussed in (Veldhuizen, 1995; Vandevoorde and Josuttis, 2003)) the saving compared to a linear combination of the $\boldsymbol{x}_i$ is small. For higher dimensions this comparison may improve, but at the same time the number of vectors $\boldsymbol{a}_i$ increases—and thereby also the additional memory overhead. Further, if one allows geometry changes, i.e., the nodes of the quadrilateral may move in such a manner that the element deforms, then the auxiliary vectors have to be recomputed, hence reducing the savings in computing time. The current compiler switch for some mappings to use a Horner-like scheme as in (B.4) should therefore be used with care and might be abolished in the future.

Integration rules

The integration rules that are implemented for the `ReferenceSquare` are listed in Table B.5.

class name	order	# points	remark
`Cn2_1_1`	1	1	centroid formula, p. 229
`Cn2_3_4`	3	4	product Gauß formula, p. 230
`Cn2_5_9`	5	9	product Gauß formula, p. 234
`C2_7_4`	7	16	product Gauß formula, p. 255

Table B.5: Integration rules related to `ReferenceSquare`.

B.3 Three-dimensional reference geometries

B.3.1 `ReferenceTetrahedron`

`ReferenceTetrahedron` defines the three-dimensional shape that corresponds to the triangle in two dimensions: it has the minimum number of vertices necessary to span the complete space with the vectors defined by its edges.

Geometry and topology

The vertex positions have been described previously and are written out in Table B.6a on the following page. The faces are defined in Table B.6b, and—additionally in three-dimensional geometries—the edges of the `ReferenceTetrahedron` are enumerated based on the specification of the connected vertices, cf. Table B.6c.

i	$\hat{\boldsymbol{x}}_i$
0	$(0,0,0)$
1	$(1,0,0)$
2	$(0,1,0)$
3	$(0,0,1)$

(a)

face #	$(i_1,\ldots)$: $\hat{\boldsymbol{x}}_i \in \mathcal{S}$			sign	remark
0	0	3	2	+1	$\hat{x}_0 = 0$
1	0	1	3	+1	$\hat{x}_1 = 0$
2	0	2	1	+1	$\hat{x}_2 = 0$
3	1	2	3	+1	$\hat{x}_0 + \hat{x}_1 + \hat{x}_2 = 1$

(b)

edge #	(i_1,i_2): $\hat{\boldsymbol{x}}_i \in$ edge	
0	0	1
1	0	2
2	0	3
3	2	3
4	1	3
5	1	2

(c)

Table B.6: `ReferenceTetrahedron` definitions, (a) vertex coordinates, (b) faces, (c) edges.

Mapping

The mapping for tetrahedra is the three-dimensional instance of the general d-simplex mapping in (B.2).

Integration rules

The integration rules that are implemented for the `ReferenceTetrahedron` are listed in Table B.7.

class name	order	# points	remark
`Tn3_1_1`	1	1	midpoint formula, p. 307
`Tn3_2_1`	2	4	p. 307
`Tn3_3_1`	3	5	negative weight, p. 308
`T3_5_1`	5	15	p. 315
`T3_7_1`	7	64	conical product Gauß formula, p. 315

Table B.7: Integration rules related to `ReferenceTetrahedron`.

B.3.2 `ReferencePyramid`

Geometry and topology

`ReferencePyramid` has a base of the shape of `ReferenceSquare` in the plane defined by $\hat{x}_2 = 0$ and its tip at $\hat{x} = (0,0,1)$. With this choice, the integration rules can be given as conical product of a rule for the reference square and one for a reference line (scaled to unit length). The vertex positions, faces and edge definitions are compiled in Table B.8.

i	$\hat{\boldsymbol{x}}_i$
0	$(0, 0, 1)$
1	$(-1, -1, 0)$
2	$(1, -1, 0)$
3	$(-1, 1, 0)$
4	$(1, 1, 0)$

(a)

face #	$(i_1, \ldots)$: $\hat{\boldsymbol{x}}_i \in \mathcal{S}$				sign	remark
0	3	4	1	2	+1	$\hat{x}_2 = 0$
1	3	1	0		+1	
2	2	4	0		+1	
3	1	2	0		+1	
4	4	3	0		+1	

(b)

edge #	(i_1,i_2): $\hat{\boldsymbol{x}}_i \in$ edge	
0	0	1
1	0	2
2	0	3
3	0	4
4	1	2
5	2	4
6	4	3
7	3	1

(c)

Table B.8: `ReferencePyramid` definitions, (a) vertex coordinates, (b) faces, (c) edges.

Mapping

For five vertices, the ansatz $G(\hat{\boldsymbol{x}}) = \boldsymbol{a} + \hat{x}_0\,\boldsymbol{a}_0 + \hat{x}_1\,\boldsymbol{a}_1 + \hat{x}_0\hat{x}_1\,\boldsymbol{a}_{01} + \hat{x}_2\,\boldsymbol{a}_2$ combines the bilinear mapping in planes parallel to the pyramid base with a linear term in the perpendicular direction and leads to the vector coefficients

$$\boldsymbol{a} = \tfrac{1}{8}(4\boldsymbol{x}_0 + \boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3 + \boldsymbol{x}_4)\,, \quad \text{(B.5a)} \qquad \boldsymbol{a}_0 = \tfrac{1}{4}(-\boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3 + \boldsymbol{x}_4)\,, \quad \text{(B.5b)}$$

$$\boldsymbol{a}_1 = \tfrac{1}{4}(-\boldsymbol{x}_1 - \boldsymbol{x}_2 + \boldsymbol{x}_3 + \boldsymbol{x}_4)\,, \quad \text{(B.5c)} \qquad \boldsymbol{a}_{01} = \tfrac{1}{4}(\boldsymbol{x}_1 - \boldsymbol{x}_2 - \boldsymbol{x}_3 + \boldsymbol{x}_4)\,, \quad \text{(B.5d)}$$

$$\boldsymbol{a}_2 = \tfrac{1}{8}(4\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2 - \boldsymbol{x}_3 - \boldsymbol{x}_4)\,. \quad \text{(B.5e)}$$

Alternatively, the above formulae can be regrouped with respect to the vertex vectors $\boldsymbol{x}_i$ so that no auxiliary vectors $\boldsymbol{a}$ need to be stored. In this case, the scalar coefficients $\xi_i(\hat{\boldsymbol{x}})$,

$$\xi_0 = \hat{x}_2\,, \quad \text{(B.6a)} \qquad \xi_1 = \tfrac{1}{4}\,(1 - \hat{x}_0 - \hat{x}_1 - \hat{x}_2 + \hat{x}_0\hat{x}_1)\,, \quad \text{(B.6b)}$$

$$\xi_2 = \tfrac{1}{4}\,(1 + \hat{x}_0 - \hat{x}_1 - \hat{x}_2 - \hat{x}_0\hat{x}_1)\,, \quad \text{(B.6c)} \qquad \xi_3 = \tfrac{1}{4}\,(1 - \hat{x}_0 + \hat{x}_1 - \hat{x}_2 - \hat{x}_0\hat{x}_1)\,, \quad \text{(B.6d)}$$

$$\xi_4 = \tfrac{1}{4}\,(1 + \hat{x}_0 + \hat{x}_1 - \hat{x}_2 + \hat{x}_0\hat{x}_1)\,, \quad \text{(B.6e)}$$

occur in the mapping, which reads

$$G(\hat{\boldsymbol{x}}) = \sum_{i=0}^{4} \xi_i(\hat{\boldsymbol{x}})\,\boldsymbol{x}_i\,. \quad \text{(B.7)}$$

Integration rules

Stroud (1971, Sec. 2.5) describes a method to generate integration rules for d-dimensional cones. For the vertical direction, the special rule to approximate integrals in the form

$$\int_0^1 (1-t)^d f(t)\,\mathrm{d}t \approx \sum_{i=1}^{n} C_i f(t_i)\,, \tag{B.8}$$

with the weights C_i and abscissae t_i is used. Such rules are given by Stroud for the base dimension $d = 1$ on page 314 and for $d = 2$ on pages 315 and 339. Since these are seventh order rules, all derived rules are seventh order accurate in the vertical direction and the numbers of integration points are multiples of four. The integration rules implemented for the reference pyramid are listed in Table B.9.

class name	order	# points	remark
`Pyramid_1_1`	1	4	conical product formula (with `Cn2_1_1`)
`Pyramid_3_1`	3	16	conical product formula (with `Cn2_3_4`)
`Pyramid_5_1`	5	36	conical product formula (with `Cn2_5_9`)
`Pyramid_7_1`	7	48	conical product formula (with `C2_7_1`, p. 252, which is not listed for the square, but used here to reduce the number of integration points from 64 to 48)

Table B.9: Integration rules related to `ReferencePyramid`.

B.3.3 `ReferenceTriangularPrism`

Geometry and topology

The base and top of the reference triangular prism are triangles of shape reference triangle at $\hat{x}_2 = -1$ and $\hat{x}_2 = +1$, respectively. Hence the integration rules can be constructed as tensor product of rules associated with `ReferenceTriangle` and `ReferenceLine`. The definition of `ReferenceTriangularPrism` is given in Table B.10 on the facing page.

Mapping

The ansatz

$$G(\hat{\boldsymbol{x}}) = \boldsymbol{a} + \hat{x}_0\,\boldsymbol{a}_0 + \hat{x}_1\,\boldsymbol{a}_1 + \hat{x}_2\,\boldsymbol{a}_2 + \hat{x}_0\hat{x}_2\,\boldsymbol{a}_{02} + \hat{x}_1\hat{x}_2\,\boldsymbol{a}_{12}\,, \tag{B.9}$$

i	$\hat{\boldsymbol{x}}_i$
0	$(0,0,-1)$
1	$(1,0,-1)$
2	$(0,1,-1)$
3	$(0,0,\ 1)$
4	$(1,0,\ 1)$
5	$(0,1,\ 1)$

(a)

face #	$(i_1,\ldots)$: $\hat{\boldsymbol{x}}_i \in \mathcal{S}$				sign	remark
0	0	2	1		+1	bottom
1	3	4	5		+1	top
2	2	0	5	3	+1	$\hat{x}_0 = 0$
3	0	1	3	4	+1	$\hat{x}_1 = 0$
4	1	2	4	5	+1	$\hat{x}_0 + \hat{x}_1 = 1$

(b)

edge #	(i_1,i_2): $\hat{\boldsymbol{x}}_i \in$ edge	
0	0	1
1	0	2
2	1	2
3	3	4
4	3	5
5	4	5
6	0	3
7	1	4
8	2	5

(c)

Table B.10: `ReferenceTriangularPrism` definitions, (a) vertex coordinates, (b) faces, (c) edges.

leads to the (vector) coefficients

$$\boldsymbol{a} = \tfrac{1}{2}(+\boldsymbol{x}_0 + \boldsymbol{x}_3)\,, \quad \text{(B.10a)} \qquad \boldsymbol{a}_0 = \tfrac{1}{2}(-\boldsymbol{x}_0 + \boldsymbol{x}_1 - \boldsymbol{x}_3 + \boldsymbol{x}_4)\,, \quad \text{(B.10b)}$$

$$\boldsymbol{a}_1 = \tfrac{1}{2}(-\boldsymbol{x}_0 + \boldsymbol{x}_2 - \boldsymbol{x}_3 + \boldsymbol{x}_5)\,, \quad \text{(B.10c)} \qquad \boldsymbol{a}_2 = \tfrac{1}{2}(-\boldsymbol{x}_0 + \boldsymbol{x}_3)\,, \quad \text{(B.10d)}$$

$$\boldsymbol{a}_{02} = \tfrac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_3 + \boldsymbol{x}_4)\,, \quad \text{(B.10e)} \qquad \boldsymbol{a}_{12} = \tfrac{1}{2}(\boldsymbol{x}_0 - \boldsymbol{x}_2 - \boldsymbol{x}_3 + \boldsymbol{x}_5)\,. \quad \text{(B.10f)}$$

Also here, the rearrangement to a linear combination of vertex vectors is given:

$$\xi_0 = \tfrac{1}{2}(1 - \hat{x}_2)(1 - \hat{x}_0 - \hat{x}_1)\,, \quad \text{(B.11a)} \qquad \xi_1 = \tfrac{1}{2}\hat{x}_0(1 - \hat{x}_2)\,, \quad \text{(B.11b)}$$

$$\xi_2 = \tfrac{1}{2}\hat{x}_1(1 - \hat{x}_2)\,, \quad \text{(B.11c)} \qquad \xi_3 = \tfrac{1}{2}(1 + \hat{x}_2)(1 - \hat{x}_0 - \hat{x}_1)\,, \quad \text{(B.11d)}$$

$$\xi_4 = \tfrac{1}{2}\hat{x}_0(1 + \hat{x}_2)\,, \quad \text{(B.11e)} \qquad \xi_5 = \tfrac{1}{2}\hat{x}_1(1 + \hat{x}_2)\,. \quad \text{(B.11f)}$$

Using these definitions, the transformation from reference to physical space is given by

$$G(\hat{\boldsymbol{x}}) = \sum_{i=0}^{5} \xi_i(\hat{\boldsymbol{x}})\,\boldsymbol{x}_i\,. \tag{B.12}$$

Integration rules

The integration rules that are implemented for the `ReferenceTriangularPrism` are listed Table B.11 on the next page.

class name	order	# points	remark
`TriPrism_1_1`	1	1	product of `Tn2_1_1` and `Cn1_1_1`
`TriPrism_3_1`	3	8	product of `Tn2_3_1` and `Cn1_3_4`
`TriPrism_5_1`	5	21	product of `Tn2_5_1` and `Cn1_5_9`
`TriPrism_7_1`	7	64	product of `T2_7_1` and `Cn1_7_x`

Table B.11: Integration rules related to `ReferenceTriangularPrism`.

B.3.4 `ReferenceCube`

Geometry and topology

`ReferenceCube` formally is a Cartesian product of `ReferenceLines` in the same way as `ReferenceSquare`. Hence, except for the edge and face definitions, the setup of this geometry is foreseeable, cf. Table B.12. The enumeration of the vertices is also depicted in Figure B.1 on the facing page.

i	$\hat{\boldsymbol{x}}_i$
0	$(-1,-1,-1)$
1	$(+1,-1,-1)$
2	$(-1,+1,-1)$
3	$(+1,+1,-1)$
4	$(-1,-1,+1)$
5	$(+1,-1,+1)$
6	$(-1,+1,+1)$
7	$(+1,+1,+1)$

(a)

face #	$(i_1,\ldots)$: $\hat{\boldsymbol{x}}_i \in \mathcal{S}$				sign	remark
0	0	1	2	3	-1	$\hat{x}_2 = -1$
1	0	1	4	5	$+1$	$\hat{x}_1 = -1$
2	0	2	4	6	-1	$\hat{x}_0 = -1$
3	1	3	5	7	$+1$	$\hat{x}_0 = +1$
4	2	3	6	7	-1	$\hat{x}_1 = +1$
5	4	5	6	7	$+1$	$\hat{x}_2 = +1$

(b)

edge #	(i_1,i_2): $\hat{\boldsymbol{x}}_i \in$ edge	
0	0	1
1	2	3
2	4	5
3	6	7
4	0	2
5	1	3
6	4	6
7	5	7
8	0	4
9	1	5
10	2	6
11	3	7

(c)

Table B.12: `ReferenceCube` definitions, (a) vertex coordinates, (b) faces, (c) edges.

Mapping

For the mapping from the `ReferenceCube` to an arbitrary hexahedral with the vertices $\boldsymbol{x}_i, i = 0,\ldots,7$, the ansatz is

$$G(\hat{\boldsymbol{x}}) = \boldsymbol{a} + \hat{x}_0\, \boldsymbol{a}_0 + \hat{x}_1\, \boldsymbol{a}_1 + \hat{x}_2\, \boldsymbol{a}_2 + \hat{x}_0\hat{x}_1\, \boldsymbol{a}_{01} + \hat{x}_0\hat{x}_2\boldsymbol{a}_{02} + \hat{x}_1\hat{x}_2\, \boldsymbol{a}_{12} + \hat{x}_0\hat{x}_1\hat{x}_2\, \boldsymbol{a}_{012}\,. \quad \text{(B.13)}$$

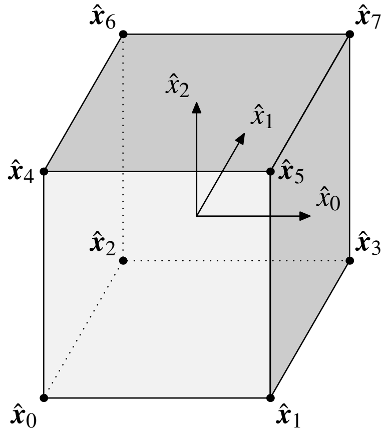


Figure B.1: Vertex enumeration defined in `ReferenceCube`.

The involved vectors are

$$\boldsymbol{a} = \tfrac{1}{8}(+\boldsymbol{x}_0 + \boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3 + \boldsymbol{x}_4 + \boldsymbol{x}_5 + \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14a}$$
$$\boldsymbol{a}_0 = \tfrac{1}{8}(-\boldsymbol{x}_0 + \boldsymbol{x}_1 - \boldsymbol{x}_2 + \boldsymbol{x}_3 - \boldsymbol{x}_4 + \boldsymbol{x}_5 - \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14b}$$
$$\boldsymbol{a}_1 = \tfrac{1}{8}(-\boldsymbol{x}_0 - \boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3 - \boldsymbol{x}_4 - \boldsymbol{x}_5 + \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14c}$$
$$\boldsymbol{a}_2 = \tfrac{1}{8}(-\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2 - \boldsymbol{x}_3 + \boldsymbol{x}_4 + \boldsymbol{x}_5 + \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14d}$$
$$\boldsymbol{a}_{01} = \tfrac{1}{8}(+\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2 + \boldsymbol{x}_3 + \boldsymbol{x}_4 - \boldsymbol{x}_5 - \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14e}$$
$$\boldsymbol{a}_{02} = \tfrac{1}{8}(+\boldsymbol{x}_0 - \boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3 - \boldsymbol{x}_4 + \boldsymbol{x}_5 - \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14f}$$
$$\boldsymbol{a}_{12} = \tfrac{1}{8}(+\boldsymbol{x}_0 + \boldsymbol{x}_1 - \boldsymbol{x}_2 - \boldsymbol{x}_3 - \boldsymbol{x}_4 - \boldsymbol{x}_5 + \boldsymbol{x}_6 + \boldsymbol{x}_7)\,, \tag{B.14g}$$
$$\boldsymbol{a}_{012} = \tfrac{1}{8}(-\boldsymbol{x}_0 + \boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{x}_3 + \boldsymbol{x}_4 - \boldsymbol{x}_5 - \boldsymbol{x}_6 + \boldsymbol{x}_7)\,. \tag{B.14h}$$

However, owing to the symmetry of the reference cube, the mapping is also easily constructed in nodal form: the functions

$$\xi_0 = \tfrac{1}{8}(1-\hat{x}_0)(1-\hat{x}_1)(1-\hat{x}_2)\,, \quad \text{(B.15a)} \qquad \xi_1 = \tfrac{1}{8}(1+\hat{x}_0)(1-\hat{x}_1)(1-\hat{x}_2)\,, \quad \text{(B.15b)}$$
$$\xi_2 = \tfrac{1}{8}(1-\hat{x}_0)(1+\hat{x}_1)(1-\hat{x}_2)\,, \quad \text{(B.15c)} \qquad \xi_3 = \tfrac{1}{8}(1+\hat{x}_0)(1+\hat{x}_1)(1-\hat{x}_2)\,, \quad \text{(B.15d)}$$
$$\xi_4 = \tfrac{1}{8}(1-\hat{x}_0)(1-\hat{x}_1)(1+\hat{x}_2)\,, \quad \text{(B.15e)} \qquad \xi_5 = \tfrac{1}{8}(1+\hat{x}_0)(1-\hat{x}_1)(1+\hat{x}_2)\,, \quad \text{(B.15f)}$$
$$\xi_6 = \tfrac{1}{8}(1-\hat{x}_0)(1+\hat{x}_1)(1+\hat{x}_2)\,, \quad \text{(B.15g)} \qquad \xi_7 = \tfrac{1}{8}(1+\hat{x}_0)(1+\hat{x}_1)(1+\hat{x}_2)\,, \quad \text{(B.15h)}$$

enter the mapping as

$$G(\hat{\boldsymbol{x}}) = \sum_{i=0}^{7} \xi_i\, \boldsymbol{x}_i\,. \tag{B.16}$$

Integration rules

The integration rules that are implemented for the `ReferenceCube` are listed in Table B.13.

class name	order	# points	remark
`Cn3_1_1`	1	1	centroid formula, p. 229
`Cn3_3_4`	3	8	product Gauß formula, p. 230
`Cn3_5_9`	5	27	product Gauß formula, p. 234
`C3_7_2`	7	34	p. 265

Table B.13: Integration rules related to `ReferenceCube`.

Bibliography

On peut être plus fin qu'un autre,
mais non pas plus fin que tous les autres.

François de La Rochefoucauld (1613–1680), *Maximes*

Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen (1999). *LAPACK users' guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd edition.

Arnold, D. N., F. Brezzi, B. Cockburn, and L. D. Marini (2002). Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, **39**(5), pp. 1749–1779.

Atkins, P. W. and J. de Paula (2002). *Physical chemistry*. Oxford University Press, 6th edition.

Bangerth, W., R. Hartmann, and G. Kanschat (2006). `deal.II` *Differential Equations Analysis Library, Technical Reference*. `http://www.dealii.org`.

Barth, T. J. (1999). *An introduction to recent developments in theory and numerics for conservation laws*, volume 5 of *Lecture Notes in Computational Science and Engineering*, chapter Numerical methods for gasdynamic systems on unstructured meshes, pp. 195–285. Springer.

Barton, J. J. and L. R. Nackman (1994). *Scientific and engineering C++*. Addison Wesley Longman, Inc.

Bassi, F., A. Crivellini, D. Di Pietro, and S. Rebay (2006). An artificial compressibility flux for the discontinuous Galerkin solution of the incompressible Navier–Stokes equations. *J. Comput. Phys.*, **218**(2), pp. 794–815.

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

Bassi, F. and S. Rebay (1997b). High-order accurate discontinuous finite element solution of the 2D Euler equations. *J. Comput. Phys.*, **138**(2), pp. 251–285.

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

Bijl, H. and P. Wesseling (1998). A unified method for computing incompressible and compressible flows in boundary-fitted coordinates. *J. Comput. Phys.*, **141**(2), pp. 153–173.

Booch, G. (1994). *Object-oriented analysis and design with applications*. The Benjamin/Cummings Publishing Company, Inc., 2nd edition.

boost C++ libraries (2007). `http://www.boost.org/`.

Brenner, S. C. and L. R. Scott (2002). *The mathematical theory of finite element methods*. Texts in Applied Mathematics. Springer, 2nd edition.

Cargill, T. (1992). *C++ programming style*. Professional Computing Series. Addison-Wesley.

Centaursoft (2005). Centaur^{TM} Grid Generator. `http://www.centaursoft.com/`.

Chen, S. and G. D. Doolen (1998). Lattice Boltzmann method for fluid flows. *Annu. Rev. Fluid Mech.*, **30**(1), pp. 329–364.

Chorin, A. J. (1967). A numerical method for solving incompressible viscous flow problems. *J. Comput. Phys.*, **2**(1), pp. 12–26.

Ciarlet, P. G. (1978). *The finite element method for elliptic problems*. North-Holland.

Cockburn, B. (1999). *High-order methods for computational physics*, volume 9 of *Lecture Notes in Computational Science and Engineering*, chapter Discontinuous Galerkin methods for convection-dominated problems, pp. 69–224. Springer.

Cockburn, B., G. Kanschat, and D. Schötzau (2002). The local discontinuous Galerkin method in incompressible fluid flow. In H. A. Mang, F. G. Rammerstorfer, and J. Eberhardsteiner (editors) *Proceedings Fifth World Congress on Computational Mechanics*.

Cockburn, B., G. E. Karniadakis, and C.-W. Shu (editors) (2000). *Discontinous Galerkin methods. Theory, computation and applications*, volume 11 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin.

Cockburn, B. and C.-W. Shu (1998). The Runge–Kutta discontinuous Galerkin method for conservation laws V: Multidimensional systems. *J. Comput. Phys.*, **141**(2), pp. 199–224.

Darmofal, D. L., P. Moinier, and M. B. Giles (2000). Eigenmode analysis of boundary conditions for the one-dimensional preconditioned Euler equations. *J. Comput. Phys.*, **160**(1), pp. 369–384.

Dormand, J. R. (1996). *Numerical methods for differential equations*. CRC Press.

Drew, D. A. and S. L. Passman (1998). *Theory of multicomponent fluids*. Applied Mathematical Sciences, Nr. 135. Springer.

Edwards, D. A., H. Brenner, and D. T. Wasan (1991). *Interfacial transport processes and rheology*. Butterworth-Heinemann series in chemical engineering. Butterworth-Heinemann.

Franca, L. P., S. L. Frey, and T. J. R. Hughes (1992). Stabilized finite element methods: I. Application to the advective-diffusive model. *Comput. Methods Appl. Mech. Engrg.*, **95**(2), pp. 253–276.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1994). *Design patterns, elements of reusable object-oriented software*. Addison-Wesley.

Ghia, U., K. N. Ghia, and C. T. Shin (1982). High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method. *J. Comput. Phys.*, **48**(3), pp. 387–411.

Godunov, S. K. (1962). The problem of a generalized solution in the theory of quasilinear equations and in gas dynamics. *Russ. Math. Surv.*, **17**(3), pp. 145–156.

Guillard, H. and A. Murrone (2004). On the behavior of upwind schemes in the low Mach number limit: II. Godunov type schemes. *Comput. & Fluids*, **33**(4), pp. 655–675.

Guillard, H. and C. Viozat (1999). On the behaviour of upwind schemes in the low Mach number limit. *Comput. & Fluids*, **28**(1), pp. 63–86.

Hairer, W., S. P. Nørsett, and G. Wanner (1993). *Solving ordinary differential equations*, volume I – Nonstiff problems. Springer, 2nd edition.

Hairer, W. and G. Wanner (1993). *Solving ordinary differential equations*, volume II – Stiff and differential-algebraic problems. Springer, 2nd edition.

Hartmann, R. (2006). Adaptive discontinuous Galerkin methods with shock-capturing for the compressible Navier–Stokes equations. *Internat. J. Numer. Methods Fluids*, **51**(9–10), pp. 1131–1156.

Hartmann, R. and P. Houston (2006). Symmetric interior penalty DG methods for the compressible Navier–Stokes equations I: Method formulation. *Int. J. Numer. Anal. Model.*, **3**(1), pp. 1–20.

Hauke, G. (2001). Simple stabilizing matrices for the computation of compressible flows in primitive variables. *Comput. Methods Appl. Mech. Engrg.*, **190**(51–52), pp. 6881–6893.

Hauke, G. and T. J. R. Hughes (1998). A comparative study of different sets of variables for solving compressible and incompressible flows. *Comput. Methods Appl. Mech. Engrg.*, **153**, pp. 1–44.

Hoffman, J. and A. Logg (2002). DOLFIN: Dynamic object oriented library for finite element computation. Preprint 2002-06, Department of Computational Mathematics, Chalmers University of Technology.

Houston, P., M. Jensen, and E. Süli (2002). hp-discontinuous Galerkin finite element methods with least-squares stabilization. *J. Sci. Comput.*, **17**(1/4), pp. 3–25.

Houston, P., D. Schötzau, and T. P. Wihler (2006). An hp-adaptive mixed discontinuous Galerkin FEM for nearly incompressible linear elasticity. *Comput. Methods Appl. Mech. Engrg.*, **195**(25-28), pp. 3224–3246.

hpGEM (2007). Discontinuous Galerkin finite element method framework, available from `http://wwwhome.math.utwente.nl/~hpgemdev`.

Hughes, T. J. R., L. P. Franca, and M. Mallet (1986). A new finite element formulation for computational fluid dynamics: I. Symmetric forms of the compressible Euler and Navier–Stokes equations and the second law of thermodynamics. *Comput. Methods Appl. Mech. Engrg.*, **54**(2), pp. 223–234.

Hundsdorfer, W. and J. Verwer (2003). *Numerical solution of time-dependent advection-diffusion-reaction equations*. Springer.

Klaij, C. M., M. H. van Raalte, J. J. W. van der Vegt, and H. van der Ven (2007). *h*-Multigrid for space-time discontinuous Galerkin discretizations of the compressible Navier–Stokes equations. Submitted to *J. Comput. Phys.*

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

Klaij, C. M., J. J. W. van der Vegt, and H. van der Ven (2006b). Space-time discontinuous Galerkin method for the compressible Navier–Stokes equations. *J. Comput. Phys.*, **217**(2), pp. 589–611.

Kleb, W. L., W. A. Wood, and B. van Leer (1999). Efficient multi-stage time marching for viscous flows via local preconditioning. In *Collection of technical papers*, number 99–3267 in 14th Computational Fluid Dynamics Conference, Norfolk, VA, June 28–July 1, 1999, pp. 1–14. American Institute of Aeronautics and Astronautics (AIAA).

Landau, L. D. and E. M. Lifshitz (1963). *Fluid mechanics*, volume 6 of *Course of Theoretical Physics*. Pergamon Press, 2nd edition.

LeVeque, R. J. (2002). *Finite volume methods for hyperbolic problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press.

Melson, N. D., H. L. Atkins, and M. D. Sanetrik (1993). Time-accurate Navier–Stokes calculations with multigrid acceleration. Technical report, NASA Langley Research Center.

Menikoff, R. and B. J. Plohr (1989). The Riemann problem for fluid flow of real materials. *Rev. Mod. Phys.*, **61**(1), pp. 75–130.

Mozolevski, I., E. Süli, and P. R. Bösing (2007). Discontinuous Galerkin finite element approximation of the two-dimensional Navier–Stokes equations in stream-function formulation. *Commun. Numer. Meth. Engrg.*, **23**, pp. 447–459.

Myers, N. C. (1995). Traits: a new and useful template technique. *C++ Report*, **7**(5), pp. 32–35.

Object Management Group (2007). Unified Modeling Language: Infrastructure. Technical Report version 2.1.1.

Panton, R. L. (1984). *Incompressible flow*. John Wiley & Sons, Inc.

Persson, P.-O. and J. Peraire (2006). Sub-cell shock capturing for discontinuous Galerkin methods. In *Proc. 44th AIAA Aerospace Sciences Meeting*, AIAA-2006-0112. Reno, Nevada.

Pesch, L., A. Bell, W. E. H. Sollie, V. R. Ambati, O. Bokhove, and J. J. W. van der Vegt (2007). hpGEM – A software framework for discontinuous Galerkin finite element methods. *ACM Trans. Math. Software*, **33**(4).

Pesch, L. and J. J. W. van der Vegt (2006). A space-time discontinuous Galerkin finite-element discretization of the euler equations using entropy variables. In P. Wesseling, E. Oñate, and J. Periaux (editors) *Proceedings of the European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2006, Egmond aan Zee, The Netherlands*. TU Delft, Delft.

Pesch, L. and J. J. W. van der Vegt (2007). A discontinuous Galerkin finite element discretization of the Euler equations for compressible and incompressible fluids. Submitted to *J. Comput. Phys.*

Polner, M. (2005). *Galerkin least-squares stabilization operators for the Navier–Stokes equations – A unified approach*. Ph.D. thesis, Department of Applied Mathematics, University of Twente.

Polner, M., L. Pesch, and J. J. W. van der Vegt (2007). Construction of stabilization operators for Galerkin least-squares discretizations of compressible and incompressible flows. *Comput. Methods Appl. Mech. Engrg.*, **196**(21–24), pp. 2431–2448.

Polner, M., J. J. W. van der Vegt, and R. M. J. van Damme (2006). Analysis of stabilization operators for Galerkin least-squares discretizations of the incompressible Navier–Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, **195**(9–12), pp. 982–1006.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen (1991). *Object-oriented modeling and design*. Prentice Hall.

Schmidt, A. and K. G. Siebert (2006). ALBERTA – An adaptive hierarchical finite element toolbox. `http://www.alberta-fem.de/`.

Sengers, J. V. (2000). *Equations of state for fluids and fluid mixtures*. Experimental thermodynamics. Elsevier.

Shakib, F., T. J. R. Hughes, and Z. Johan (1991). A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier–Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, **89**(1–3), pp. 141–219.

Soh, W. Y. and J. W. Goodrich (1988). Unsteady solution of incompressible Navier–Stokes equations. *J. Comput. Phys.*, **79**, pp. 113–134.

Sollie, W. E. H., J. J. W. van der Vegt, and O. Bokhove (2007). A space-time discontinuous Galerkin finite element method for two-fluid flow problems. TW-Memorandum 1849, see `http://www.math.utwente.nl/publications/`.

Stroud, A. H. (1971). *Approximate calculation of multiple integrals*. Prentice-Hall.

Tadmor, E. (2003). Entropy stability theory for difference approximations of nonlinear conservation laws and related time-dependent problems. *Acta Numer.*, **12**, pp. 451–512.

Tecplot, Inc. (2005). Tecplot. `http://www.tecplot.com/`.

Toro, E. F. (1989). A fast Riemann solver with constant covolume applied to the random choice method. *Internat. J. Numer. Methods Fluids*, **9**, pp. 1145–1164.

Toro, E. F. (1999). *Riemann solvers and numerical methods for fluid dynamics: A practical introduction*. Springer, 2nd edition.

Toro, E. F., M. Spruce, and W. Speares (1994). Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves*, **4**(1), pp. 25–34.

Turkel, E. (1999). Preconditioning techniques in computational fluid dynamics. *Annu. Rev. Fluid Mech.*, **31**(1), pp. 385–416.

Turkel, E., R. Radespiel, and N. Kroll (1997). Assessment of preconditioning methods for multidimensional aerodynamics. *Comput. & Fluids*, **26**(6), pp. 613–634.

Vandevoorde, D. and N. M. Josuttis (2003). *C++ templates – The complete guide*. Addison-Wesley.

van der Vegt, J. J. W. (2006). Personal communication.

van der Vegt, J. J. W. and S. K. Tomar (2005). Discontinuous Galerkin method for linear free-surface gravity waves. *J. Sci. Comput.*, **22**(1), pp. 531–567.

van der Vegt, J. J. W. and H. van der Ven (2002a). Slip flow boundary conditions in discontinuous Galerkin discretizations of the Euler equations of gas dynamics. In H. A. Mang, F. G. Rammerstorfer, and J. Eberhardsteiner (editors) *Proceedings Fifth World Congress on Computational Mechanics*, pp. 1–16.

van der Vegt, J. J. W. and H. van der Ven (2002b). Space-time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows: I. General formulation. *J. Comput. Phys.*, **182**(2), pp. 546–585.

Veldhuizen, T. (1995). Using C++ template metaprograms. *C++ Report*, **7**(4), pp. 36–43.

van der Ven, H. and J. J. W. van der Vegt (2002). Space-time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows II. Efficient flux quadrature. *Comput. Methods Appl. Mech. Engrg.*, **191**(41–42), pp. 4747–4780.

Warsi, Z. U. A. (1999). *Fluid dynamics: Theoretical and computational approaches*. CRC Press, 2nd edition.

Wesseling, P. (2000). *Principles of computational fluid dynamics*. Springer.

Westhuis, J.-H. (2001). *The numerical simulation of nonlinear waves in a hydrodynamic model test basin*. Ph.D. thesis, Faculty of Mathematical Sciences, University of Twente.

Wikipedia (2007). Object-oriented programming — wikipedia, the free encyclopedia. Online; accessed 10-April-2007.

Zdunkowski, W. and A. Bott (2004). *Thermodynamics of the atmosphere*. Cambridge University Press.

Summary

This thesis presents numerical methods and tools for the simulation of the flow of fluids with different thermodynamical properties. Against the background of gas-liquid multiphase flow, which is frequently simulated by simplistically assuming both components to be incompressible, the presentation runs along the lines of three problems that have to be solved to improve the description and simulation tools for multiphase flow.

First, a mathematical model is sought that allows the numerical solution for both compressible and incompressible fluids. When considering the Navier–Stokes equations in terms of conservation variables, the incompressible limit is singular. By solving for certain generalized variable sets rather than conservation variables, the singularity can be avoided. Such variable sets with a well-defined incompressible limit are pressure primitive variables and entropy variables. The different sets are introduced and the advantages and disadvantages of numerical methods based on them are summarized. Especially entropy variables, which stem from symmetrization theory, give rise to desirable properties like automatic fulfillment of the second law of thermodynamics.

To test the generalized variable formulation, a time discontinuous Galerkin least-squares finite element discretization of the Navier–Stokes equations is presented. An essential component to ensure stability of the method is the least-squares operator. A recently devised stabilization matrix applicable to compressible and incompressible flow is evaluated. With this ingredient, the discretization is suitable to simulate the flow of fluids with widely differing thermodynamic behavior. However, the second central question asks for more geometric flexibility than the least-squares method with spatially continuous basis functions can offer. This is the reason to focus efforts on developing a space-time discontinuous Galerkin finite element method. This kind of discretization allows local adaptation and leads to minimal inter-element coupling, which is beneficial for computational purposes. A method previously applied to compressible flow with an ideal gas equation of state is extended with the entropy variable formulation for use with general fluids. Particular attention is paid to the examination which components of the numerical method have to be changed or adapted when using different thermodynamical models. Different possibilities of solving the nonlinear algebraic system by pseudo-time integration are investigated. The discretization of the Euler equations is augmented by the viscous terms of the Navier–Stokes equations using the interior penalty method. Numerical results for a diverse range of compressible and incompressible test cases underline the applicability of the method for various fluids and conditions.

The numerical methods discussed in this dissertation lead to complex solution algorithms that have to be implemented as software to be applied in practice. The third central topic and an important part of the current project is the development of hpGEM, a general-purpose discontinuous Galerkin finite element software framework. hpGEM provides implementations of common data structures and functionality necessary to translate discontinuous Galerkin discretizations for a wide range of partial differential equation problems into computer programs. It facilitates and accelerates the implementation of finite element methods, the assessment of algorithms, and their application. This dissertation presents the status of the framework, exemplifies aspects of its philosophy and design, and discusses several examples of how the components can be applied.

The combination of the formulation in terms of general sets of variables, a mathematical method that accommodates geometric flexibility and adaptivity, and a software environment for the implementation of numerical methods advances several steps towards the application of finite element methods to gas-liquid multiphase flows.

Samenvatting

Dit proefschrift behandelt numerieke methoden voor de simulatie van de stroming van gassen en vloeistoffen met uiteenlopende thermodynamische eigenschappen. De ontwikkelde technieken zijn bedoeld voor een betere beschrijving van gas-vloeistof-meerfasenstromingen. Deze worden tot dusver meestal als incompressibel beschouwd, terwijl dit voor de gas fractie vaak een grove vereenvoudiging is. Drie onderwerpen staan centraal in het verrichtte onderzoek en de presentatie van de resultaten.

Ten eerste wordt een wiskundig model gezocht dat voor zowel compressibele als incompressibele vloeistoffen een correct gedefinieerd probleem oplevert. Het gebruik van de standaard conservatieve grootheden bij het oplossen van de Navier–Stokes vergelijkingen leidt echter tot een singuliere incompressibele limiet. Daarentegen kan men door gebruik te maken van andere variabelen deze singulariteit vermijden. Variabelen met een correct gedefinieerde incompressibele limiet zijn de zogenaamde primitieve variabelen en de entropievariabelen. De verschillende variabelen worden geïntroduceerd en een overzicht van de voor- en nadelen van daarop gebaseerde numerieke methoden wordt gegeven. Met name de entropievariabelen, die uit de symmetriseringstheorie voortkomen, bieden aantrekkelijke eigenschappen, bijvoorbeeld het automatisch voldoen aan de tweede wet van de thermodynamica. Om de formulering met generieke variabelen te testen wordt een tijdsdiscontinue Galerkin kleinste-kwadraten eindige elementen discretisatie gebruikt. Cruciaal voor de stabiliteit van deze methode is de kleinste-kwadraten operator. Een recentelijk voorgestelde stabilisatiematrix die zowel voor compressibele als incompressibele stromingsproblemen geschikt is wordt geëvalueerd. Met deze operator is de methode in staat om vloeistoffen met verschillend thermodynamisch gedrag te simuleren.

Terwijl de ruimtelijk continue Galerkin methode de simulatie van zowel compressibele als incompressibele media mogelijk maakt laat hij wat de geometrische flexibiliteit van de discretisatie betreft te wensen over. Op dit gebied heeft de methode enkele tekortkomingen aangezien de ruimtelijke basisfuncties continu moeten zijn. Daarom wordt in het tweede deel een ruimte-tijdsdiscontinue Galerkin methode toegepast. Dit soort discretisatie is goed te combineren met lokale roosterverfijning en levert minimale koppeling tussen de vrijheidsgraden van verschillende elementen op. Dit is vooral voor de behandeling op parallele computersystemen voordelig. Een methode die eerder voor niet-viskeuze compressibele stromingsproblemen met een ideaal gas toestandsvergelijking gebruikt werd, wordt uitgebreid met de entropievariabelen formulering zodat ook andere vloeistoffen gesimuleerd kunnen worden. Hierbij wordt bijzondere aandacht besteed aan het handhaven van het generieke karakter van de methode door zo weinig mogelijk aanpassingen aan verschillende typen vloeistoffen toe te laten. Verder worden verschillende manieren onderzocht om het niet-lineaire algebraïsche systeem van vergelijkingen met behulp van pseudo-tijd integratie op te lossen. De discretisatie van de Euler vergelijkingen wordt ook uitgebreid met de viskeuze termen van de Navier–Stokes vergelijkingen. Hiervoor wordt een methode met een interne strafterm gebruikt. Numerieke voorbeelden voor een aantal testgevallen met verschillende compressibele en incompressibele vloeistoffen benadrukken de veelzijdige toepasbaarheid van het resulterende schema.

De numerieke methoden welke in dit proefschrift behandeld worden leveren complexe algoritmes op, die als computerprogramma's geïmplementeerd moeten worden om praktische problemen op te lossen. Het derde centrale thema van dit proefschrift is de ontwikkeling van hpGEM, een generiek platform voor de implementatie van discontinue Galerkin methoden. hpGEM stelt een aantal datastructuren en procedures beschikbaar, die voor de implementatie van discontinue Galerkin discretisaties voor veel verschillende typen partiële differentiaalvergelijkingen nodig zijn. Daardoor wordt de implementatiestap vereenvoudigd en versneld. De status van de ontwikkeling van hpGEM wordt gedocumenteerd en belangrijke aspecten van de onderliggende ideeën en het software design uiteen gezet. Ook voorbeelden van de toepassing van diverse componenten worden gegeven.

De combinatie van de formulering met gegeneraliseerde variabelen, een wiskundige methode die geometrisch flexibel is, alsmede lokale verfijning ondersteunt en een software omgeving voor de implementatie van numerieke methoden zijn belangrijke onderdelen in de constructie van een eindige elementen methode die toe te passen is op gas-vloeistof meerfasenstromingen.

Zusammenfassung

In dieser Dissertation werden numerische Methoden für die Strömungssimulation von Fluiden mit unterschiedlichen thermodynamischen Eigenschaften entwickelt. Vor dem Hintergrund von Mehrphasenströmungen mit Gas- und Flüssigkeitsphase, die häufig vereinfachend beide als inkompressibel betrachtet werden, werden drei Hauptthemen behandelt, die auf eine Verbesserung der Beschreibung von und Simulationstechniken für solche Strömungen abzielen.

Zunächst wird ein mathematisches Modell gesucht, das es erlaubt, numerische Methoden zu entwerfen, die sowohl für kompressible als auch inkompressible Fluide wohldefinierte Formulierungen ergeben. Betrachtet man die Navier-Stokes-Gleichungen in Abhängigkeit von den zu ihrer Herleitung benutzten Erhaltungsgrößen, so ist der inkompressible Grenzfall singulär. Gebraucht man stattdessen andere Variablensätze, dann kann diese Singularität vermieden werden. Variablen mit wohldefiniertem inkompressiblem Limit sind die sogenannten primitiven Variablen und die Entropievariablen. Diese Variablengruppen werden zunächst eingeführt und die Vor- und Nachteile auf ihnen basierender numerischer Methoden zusammengefasst. Insbesondere die Entropievariablen, die aus der mathematischen Symmetrisierungstheorie resultieren, bieten interessante Eigenschaften wie z.B. die automatische Erfüllung des zweiten Hauptsatzes der Thermodynamik.

Der Gebrauch der generalisierten Variablensätze wird anhand einer Galerkin-Kleinste-Quadrate-Diskretisierung der Navier-Stokes-Gleichungen demonstriert. Essentiell für die Stabilität dieser Methode ist der Kleinste-Quadrate-Operator. Eine kürzlich vorgestellte Stabilisierungsmatrix, die sowohl für kompressible als auch inkompressible Medien anwendbar ist, wird getestet. Dank dieses Bestandteils ist die Diskretisierung geeignet, um Strömungsprobleme für thermodynamisch unterschiedliche Medien zu simulieren, womit das erste Ziel dieser Arbeit erreicht ist.

Des Weiteren wird jedoch von der numerischen Methode geometrische Flexibilität gefordert, die für die Behandlung von Problemen mit verschiedenen Skalen und freien Oberflächen benötigt wird. In dieser Hinsicht hat die zunächst betrachtete Galerkin-Methode mit räumlich kontinuierlichen Basisfunktionen wesentliche Einschränkungen. Daher wird im Weiteren eine (Raum- und Zeit-) diskontinuierliche Galerkin (DG) Finite-Elemente-Methode entwickelt. Dieser Diskretisierungstyp erlaubt eine große Klasse lokaler Verfeinerungen und führt überdies zur minimalen Kopplung der Freiheitsgrade von verschiedenen Elementen, was vor allem der Behandlung auf Parallelrechnersystemen entgegenkommt. Eine DG Methode zur Behandlung nichtviskoser Strömungen idealer Gase wird um die Entropievariablenformulierung erweitert, so dass verschiedenartige Fluide und Strömungsgegebenheiten behandelt werden können. Besondere Aufmerksamkeit gilt hierbei der weitgehenden Vermeidung von Veränderungen, um die numerische Methode an verschiedene thermodynamische Modelle anzupassen. Das Lösen des nichtlinearen algebraischen Gleichungssystems geschieht mittels Pseudo-Zeit-Integration. Unterschiedliche Anwendungsmöglichkeiten dieses Verfahrens im Zusammenhang mit dem Gebrauch von generalisierten Variablen werden betrachtet. Außerdem werden der Diskretisierung der Euler-Gleichungen die viskosen Terme der Navier-Stokes-Gleichungen unter Benutzung einer Strafterm-Methode hinzugefügt. Simulationsergebnisse für eine Vielzahl von Testfällen, die sowohl Gase als auch Flüssigkeiten betreffen, unterstreichen die Anwendbarkeit der Methode für große Klassen von Fluiden und physikalischen Bedingungen.

Die numerischen Methoden, die im Rahmen dieser Dissertation entwickelt wurden, führen zu komplexen Lösungsalgorithmen, die als Computerprogramme implementiert werden müssen, um auf praktische Probleme anwendbar zu sein. Der dritte Teil der Arbeit widmet sich diesem Arbeitsschritt mit besonderer Berücksichtigung der Vereinfachung und Beschleunigung des Implementierungsprozesses. Dazu wird die Entwicklung von hpGEM, einem universell anwendbaren Softwaresystem für DG Finite-Elemente-Methoden, beschrieben. hpGEM stellt Implementierungen von typischerweise benötigten Datenstrukturen und Funktionalitäten bereit, die es erlauben, DG Diskretisierungen für eine Vielzahl von partiellen Differentialgleichungsproblemen in verständlicher Weise als Computerprogramm umzusetzen. Der Entwicklungsstatus von hpGEM wird im Rahmen dieser Arbeit dokumentiert, Beispiele für die zugrunde liegenden Ansätze und Entwurfstechniken werden angeführt und die Anwendung der Komponenten demonstriert.

Die in dieser Arbeit vorgestellte Kombination der Formulierung mittels verallgemeinerter Variablen, einer Diskretisierung, die hohe Ansprüche hinsichtlich geometrischer Flexibilität und Adaptivität erfüllt und einer Softwareumgebung zur einfachen Implementierung der entwickelten Methoden zeigt einen vielversprechenden Weg zur Anwendung von Finite-Elemente-Methoden auf Mehrphasenströmungsprobleme auf.

Acknowledgements

This thesis is the result of a four-years research project and could not have been completed without the help and encouragement of numerous people, to whom I would like to pay tribute here.

For the past four years I have worked in the Numerical Analysis and Computational Mechanics group of Jaap van der Vegt. Jaap, you have given me the chance to work on a current topic in a (for me) new field of science. To be offered this opportunity, together with the freedom you gave me, provided a lot of motivation. Your guidance and support have influenced my professional and personal development and with your never-failing enthusiasm and optimism you have contributed vastly to both this dissertation and the hpGEM project.

A part of this thesis derives from joint work with Monika Polner. Monika, thank you for the pleasant collaboration and support. For the discontinuous Galerkin discretization and pseudo-time methods, Chris Klaij has been an experienced contact person. Chris, thanks for sharing your knowledge and insights.

I would like to thank the members and users of the hpGEM project: Alexander, Davit, Domokos, Henk, Pablo, Sander, and Vijaya. I learned a lot from our discussions about various mathematical and computer scientific problems. On several occasions, Onno Bokhove's ability to push the project ahead has been invaluable. Onno, thank you for challenging me in both science and sports.

I acknowledge the financial support from STW and the co-funding companies and thank the members of the user committee of the two-phase flow project for their feedback during the regular meetings. Special thanks go to my fellow PhD students Hanneke and Dongsheng from the cooperating groups Physics of Fluids and Fundamentals of Chemical Reaction Engineering at the University of Twente. Niels, thanks for keeping track of the administrative matters of the project.

Next, I would like to acknowledge my present and former office-mates: Janivita, Joris, Milan, Satyendra, Tim, and Yan. Thank you for all the scientific, semi-scientific, and non-scientific conversations we have had, for the pleasant office hours, and for tolerating a colleague who keeps half a cupboard's contents of sports clothes around his desk.

Further on, I appreciate the friendly atmosphere in the Mathematical Physics and Computational Mechanics group, considerably contributed to by Mariëlle and Diana, whom I also thank for their help with many administrative duties.

Also, I would like to express my sincere gratitude to the members of my graduation committee for their time to read and evaluate my thesis: dr. ir. O. Bokhove, prof. dr. H. J. H. Clercx, prof. dr. ir. H. W. M. Hoeijmakers, Prof. Dr. G. Kanschat, prof. dr. ir. J. A. M. Kuipers, prof. dr. R. M. M. Mattheij, and prof. dr. ir. J. J. W. van der Vegt. Your comments have helped to improve this dissertation.

Luckily there have been enough people to once in a while remind me that there are other things besides mathematics. Thanks go to the friends I have made here and in particular to Ming and Natanael. Of my former fellow students especially Katrin, Rene, and Su-Jung deserve mentioning; I appreciate that distances have not affected our friendship. Sven and Sabine, thanks for sharing both good and difficult times in our PhD phase and for countless other things. Sven, without your reading of the complete manuscript this thesis would contain a lot more flaws. Thank you so much! Liisa, thank you for your continued friendship.

Last but not least I would like to thank my parents and my sister Antje for their support and encouragement.

Index