Deutsches Zentrum
für Luft- und Raumfahrt e.V.

Technical Report

Technical Documentation of the DLR TAU-Code Release 2010.1.0

Institute of Aerodynamics and Flow Technology
Braunschweig

DLR **Deutsches Zentrum für Luft- und Raumfahrt** e.V.
in der Helmholtz-Gemeinschaft

Technical Documentation of the DLR TAU-Code Release 2010.1.0

Institute of Aerodynamics and Flow Technology
Braunschweig

201 Seiten
84 Bilder
1 Tabelle
81 Literaturstellen

Technical Documentation of the DLR TAU-Code Release 2010.1.0

Deutsches Zentrum
für Luft- und Raumfahrt e.V.

Abstract

The DLR TAU-Code is a development of the DLR-Institute of Aerodynamic and Flow Technology at Göttingen and Braunschweig. This document gives an overview about technical aspects.

Contents

1 Preprocessing **1**

1.1 Introduction . . . 1
1.2 The Primary Grid . . . 1
1.2.1 Check of the Volume Element Orientations . . . 3
1.2.2 Check of the Surface Element Orientations . . . 4
1.3 Primary Grid Data structures . . . 4
1.3.1 The Point to Element Connectivity . . . 4
1.3.2 The Point to Face Connectivity . . . 5
1.3.3 Mapping of the Boundary Marker . . . 5
1.3.4 The Edge/Face to Point Connectivity . . . 6
1.3.5 The Boundary Face to Point Connectivity . . . 6
1.4 The Secondary Grid . . . 6
1.4.1 Definition of the Secondary Grid Control Volumes . . . 7
1.4.2 Inner Faces . . . 11
1.4.3 Boundary Faces . . . 11
1.4.4 Computation of Face Normals and Control Volumes . . . 12
1.5 Minor Additional information . . . 13
1.5.1 Determination of the Neighboring Point . . . 13
1.5.2 Computation of the Fan Area . . . 13
1.6 Vectorization Concerns . . . 14
1.6.1 Coloring of the Inner Faces . . . 14
1.6.2 Recursive Ordering . . . 15
1.7 Multigrid Concerns . . . 15
1.7.1 Topological Fusing Part . . . 16
1.7.2 Fusing Part . . . 22
1.8 Chimera method . . . 24
1.8.1 Introduction . . . 24
1.8.2 Stencil search . . . 25
1.8.3 Calculations of shape functions. . . . 26
1.8.4 Parallel implementation of overset unstructured grids method . . . 29

2 Flow Solver **31**
2.1 Introduction . . . 31
2.2 Governing Equations . . . 32
2.3 Upwind Schemes - Spatial Discretization . . . 34
2.3.1 AUSM-Scheme . . . 36
2.3.2 AUSMDV-Scheme . . . 38
2.3.3 Higher Order Reconstruction . . . 41
2.3.4 Correction of Gradients . . . 46
2.3.5 Limitation of Gradients . . . 47
2.4 Central Scheme - Spatial Discretization . . . 49
2.4.1 Spatial Discretization of Convective Terms . . . 50
2.4.2 Artificial Dissipation . . . 51
2.4.3 Determination of Laplacians . . . 54
2.5 Solution technique for steady state problems . . . 55
2.5.1 Steady state problems . . . 55
2.5.2 Acceleration Techniques . . . 56
2.5.3 Local Time Stepping . . . 56
2.5.4 Residual Smoothing . . . 57
2.5.5 Explicit Residual Smoothing . . . 57
2.5.6 Implicit Residual Smoothing . . . 57
2.5.7 Multigrid . . . 58
2.5.8 General Idea of Multigrid . . . 58
2.5.9 Motivation: The Correction Scheme for linear problems . . . 60
2.5.10 The Full Approximation Scheme . . . 61
2.5.11 The Full Approximation Scheme for the compressible NSE . . . 63
2.5.12 Multigrid restrictions . . . 64
2.5.13 Prolongation of the Corrections . . . 65
2.5.14 Smoothing Operators . . . 65
2.5.15 Calculation of the Time Step Size . . . 65
2.5.16 Computation of the Global Residual . . . 67
2.6 Solution techniques for time-accurate computations . . . 67
2.6.1 Global time stepping scheme . . . 68
2.6.2 Dual-time stepping scheme . . . 68
2.7 Initialization of the Flow Conditions . . . 70
2.7.1 Reference Quantities . . . 70
2.7.2 Dimensionless Quantities . . . 71

3 Boundary Treatment **73**
3.1 Inviscid Wall Flux Computation . . . 74
3.2 Projection of Velocities . . . 75

3.3 Farfield Boundary Flux Computation . . . 75
3.3.1 Supersonic Inflow/Outflow . . . 76
3.3.2 Subsonic Outflow . . . 77
3.3.3 Subsonic Inflow . . . 77
3.3.4 Nacelle Inflow Flux Computation . . . 78
3.4 Nacelle Exhaust Flux Computation . . . 79
3.5 Calculation of Pressure Forces . . . 80
3.6 Calculation of Viscous Forces . . . 81
3.7 Realization of Boundary Conditions . . . 81

4 Turbulence models **83**
4.1 The turbulent equations . . . 83
4.2 Turbulence modeling in the DLR TAU-Code . . . 85
4.2.1 Preliminary relations for eddy-viscosity models . . . 85
4.3 Spalart-Allmaras type turbulence models . . . 86
4.3.1 Spalart-Allmaras model – Original version . . . 87
4.3.2 Edwards version . . . 88
4.3.3 Initial, freestream and boundary conditions . . . 88
4.3.4 Model constants . . . 88
4.3.5 Strain-adaptive Spalart-Allmaras (SALSA) model . . . 88
4.4 Two-equation k-ω models . . . 89
4.4.1 The Wilcox k-ω . . . 89
4.4.2 The Menter baseline model . . . 90
4.4.3 The Menter SST model . . . 90
4.4.4 The NLR TNT-modification of the k-ω model . . . 91
4.4.5 The Wilcox k-ω model with SST correction . . . 91
4.4.6 The Menter two-layer k-ε model . . . 92
4.4.7 LEA k-ω model by Rung . . . 92
4.5 Explicit algebraic Reynolds stress models (EARSM) . . . 93
4.5.1 Preliminary remarks on EARSM . . . 93
4.5.2 The EARSM by Wallin and Johansson . . . 93
4.5.3 The Hellsten EARSM . . . 95
4.5.4 RQEVM k-ω model by Rung . . . 96
4.6 Vortical Correction Models . . . 97
4.7 Detached-eddy simulation . . . 103
4.7.1 Spalart-Allmaras original model with DES modification . . . 103
4.7.2 Menter SST k-ω model with DES modification . . . 104
4.7.3 Extra-large eddy simulation XLES . . . 105
4.8 Scale-Adaptive Simulation Models . . . 105
4.9 Implementation details . . . 106

4.9.1 Limitation of *k* and *omega* . . . 106
4.9.2 Initial, freestream and boundary conditions . . . 106
4.10 Transition modeling . . . 107
4.11 Wall functions (Near-wall modeling) . . . 107

5 Accuracy Improvements **110**
5.1 Introduction . . . 110
5.2 Representation of preconditioners . . . 112
5.3 Spectral Analysis of Euler's equation . . . 114
5.4 Definition of preconditioners . . . 117
5.5 Parameter choice rules . . . 119
5.6 Implemented preconditioners . . . 122
5.7 Numerical examples . . . 126
5.7.1 NACA0012 . . . 126
5.8 Technical implementation details . . . 132
5.9 Concluding remarks . . . 133

6 Parallelization **134**
6.1 Introduction . . . 134
6.2 Specification of the domain decomposition . . . 134
6.2.1 Assignment to the subdomains . . . 134
6.3 Algorithmic description of the domain decomposition . . . 136
6.3.1 Required data . . . 136
6.3.2 Grid partitioning . . . 137
6.3.3 Sub-grid definition . . . 137
6.4 Extension to the multigrid case . . . 143
6.4.1 Additional grid and sub-grid data . . . 143
6.4.2 More *additional points* on the subgrids . . . 144
6.5 Communication interface . . . 144
6.5.1 Communication index fields . . . 144
6.5.2 Communication routine . . . 146
6.6 Modifications of the flow solver . . . 148
6.6.1 Loops over all points or only over the own points? . . . 148
6.6.2 Adding exchange_pointdata() calls . . . 148
6.6.3 Using global reduction operations . . . 149
6.7 Grid partitioners . . . 149
6.7.1 Simple partitioner . . . 149
6.7.2 Metis partitioner . . . 149

7 Unsteady **150**

7.1 Deforming mesh without GCL . . . 150
7.2 Deforming mesh with GCL . . . 150

8 Grid-Adaptation Indicator 151
8.1 Introduction . . . 151
8.1.1 Gradient-Based Indicator . . . 151
8.1.2 Residual-Based Indicator . . . 152
8.1.3 Reconstruction-Based Indicator . . . 153
8.1.4 Calculation of Gradients on the Primary Grid . . . 153
8.1.5 Calculation of y_+ Values . . . 154
8.1.6 Application of Adaptation Indicators for Local Grid Refinement . . . 154

9 Grid-Adaptation Algorithm 155
9.1 Introduction . . . 155
9.2 Basic Ideas . . . 155
9.2.1 Allowable Refinement Cases . . . 156
9.2.2 The List of 'Critical' Elements . . . 159
9.2.3 Dealing with Semi-Structured Parts . . . 159
9.3 Implementation . . . 160
9.3.1 Reading the Grid and the Point-Coordinates . . . 161
9.3.2 Calculating the Missing Grid Information . . . 161
9.3.3 Reading the Curve Information . . . 162
9.3.4 Reading the Critical Elements Lists . . . 163
9.3.5 Indicating Edges Which are not Allowed to be Bisected . . . 163
9.3.6 Calculating the Values of the Chosen Indicator on Each Edge . . . 163
9.3.7 Marking Edges to be Bisected . . . 164
9.3.8 Marking Additional Edges . . . 164
9.3.9 Counting the Number of New Elements . . . 165
9.3.10 Calculating the New Points . . . 166
9.3.11 Adapting the Wall Normal Pointdistribution . . . 168
9.3.12 Interpolating the Solution . . . 168
9.3.13 Constructing and Writing the Tetrahedra and the Critical Tetrahedra . . . 168
9.3.14 Constructing and Writing the Remaining New and Critical Elements . . . 170
9.3.15 Writing the Elements, Curve Edges and Point-Coordinates . . . 171

10 Periodic Boundaries and Actuator Disks in the Primary Grid of TAU 172
10.1 Introduction . . . 172
10.2 Grid Topology . . . 172
10.3 CAD Model . . . 172
10.4 NetCDF Format for Periodic Boundaries . . . 173

11 Intermesh **180**
11.1 Preface . . . 180
11.2 Blockwise interpolation of point coordinates . . . 180
11.3 Parallelization . . . 180
11.3.1 Similar domain boundaries in all pregenerated grids . . . 183
11.3.2 Different domain boundaries in pregenerated grids . . . 183

12 Grid Quality Improvement **184**
12.1 SmoothTaugrid . . . 184
12.1.1 Quality measures . . . 184
12.1.2 Goal functions . . . 186
12.2 Methods for grid improvement . . . 186
12.2.1 Edge swapping . . . 186
12.2.2 Face swapping . . . 188
12.2.3 Edge collapsing . . . 189
12.2.4 Movement of nodes . . . 190
12.2.5 Decisions for modifications . . . 190
12.2.6 General algorithm . . . 191
12.3 Configurations of equatorial edge swapping . . . 192

Latin symbols

A	Area
A	Dissipation term
A, B	Arrays to be ordered recursively
A_n, B_n	Child parts of arrays A and B
ΔA	Difference between area relations
$\vec{C}$	Corrections
$\vec{C}^c$	Corrections
C_P^F	Point to face connectivity list
$C_P^{V^P}$	Point to prismatic volume element connectivity list
$C_P^{V^T}$	Point to tetrahedral volume element connectivity list
$C_P^{P_p}$	Point to parent point connectivity list
$C_P^{P_c}$	Point to children points connectivity list
$C_{Idx}^{P_C}$	Dress of first entry in $C_P^{P_c}$
C_F^P	Face to point connectivity list
$C_{F_b}^P$	Boundary face to point connectivity list
$C_{F_b}^{P_r}$	Boundary face to reference point connectivity list
$C_{S^Q}^P$	Corner points of quadrilateral surface element
$C_{S^T}^P$	Corner points of triangular surface element
$C_{V^P}^P$	Corner points of prismatic volume element
$C_{V^T}^P$	Corner points of tetrahedral volume element
CFL	CFL number
d	Distance
$\vec{D}$	Vector of dissipative fluxes
$\vec{D}$	Defect
E	Internal energy
F	Secondary grid face
$\vec{F} = (F^x, F^y, F^z)^T$	Normal vector of face F
$\bar{\bar{F}}$	Flux Tensor
$\vec{F}, \vec{G}, \vec{H}$	Flux vectors
F_b	Boundary face
$\vec{F}^F$	Forcing Function
F_{start}	Temporary address pointer
$\vec{F}_b = (F_b^x, F_b^y, F_b^z)^T$	Normal vector of face F_b
H	Enthalpy
H_x	Helping point
$\vec{H}_x$	Physical coordinates of helping point H_x
I^c	Transfer operator for the corrections
I^r	Transfer operator for the restrictions
I^s	Seed point identifier
I^u	Transfer operator for the flow quantities
L_{na}^P	List of not agglomerated neighbors
$L_P^{P,1,2}$	Point to point temporary list
La	Laval number
M	Mach number
$M_{S^Q}^b$	Marker of boundary type for quadrilateral surface elements

$M^b_{S^T}$	Marker of boundary type for tetrahedral surface elements
N^b	Number of boundary parts
N^{nmax}	Maximal number of points that can be agglomerated
$\vec{N}_{axis}$	Direction of nacelle axis
N^F	Number of secondary grid faces
N^{F_b}	Number of boundary faces
N^P	Number of grid points
$N^P_n a$	Number of not agglomerated points
N^{S^Q}	Number of surface quadrilaterals
N^{S^T}	Number of surface triangles
N^{V^P}	Number of prismatic volume elements
N^{V^T}	Number of tetrahedral volume elements
N^{V^H}	Number of hexahedral volume elements
P	Grid point
Pr	Priority (for agglomeration)
$\vec{P} = (P^x, P^y, P^z)^T$	Physical coordinates of point P
$\vec{Q}^F$	Fluxes over control volume faces
$\vec{Q}_1, \vec{Q}_2$	List of fine grid points to be agglomerated, order w.r.t. point priorities
$\vec{Q}^{f/li}_{Idx}(Pr)$	Address of the first/last entry in $\vec{Q}_i$ for the priority Pr
$\vec{R}$	Residuum
Res	Global Residual
S_M	Mesh stretching coefficient
S^Q	Quadrilateral surface element
S^T	Triangular surface element
$\vec{S}$	Forces on a boundary
V^P	Prismatic volume element
V^T	Tetrahedral volume element
V^H	Hexahedral volume element
V	Volume
$\vec{W}$	Vector of Conservative Variables
$\vec{Z}$	Control volume stretching vector
a	Speed of sound
a_0, b_0	Pointers to the first and the last entry of the arrays A and B
a_0, b_0	Pointers to the first and the last entry of the arrays A_x and B_x
c	Color
c_d, c_l	Lift- and drag coefficient
$\vec{d}$	Vector pointing from grid point to secondary grid face
d_{Cut}	Dissipation coefficient
d^0_{Cut}	Dissipation factor
e_x, e_y, e_z	Unit vectors in coordinate directions
h	Tetrahedron
$k^{(2,4)}$	Input parameter to control the dissipation of 2nd and 4th order
$k_{P(i)}$	Local position of point $P(i)$ in the actual volume element
l	Level of recursive ordering, left state
l_{max}	Highest level of recursive ordering
l_{vec}	Computer vector length

$\vec{n}$	Outer normal vector
$n^{x,y,z}$	Normalized components of the normal vector
$n_c^F(i)$	Number of secondary grid faces of color c
$n_{na}^P(i)$	Number of not agglomerated neighbors of $P(i)$
n_{Planes}	Number of planes for boundary points
$n_P^F(i)$	Number of secondary grid faces $P(i)$ is related to
$n_P^P(i)$	Number of neighboring points ($= n_P^P(i)$)
$n_P^{V^P}(i)$	Number of prismatic volume elements $P(i)$ is related to
$n_P^{V^T}(i)$	Number of tetrahedral volume elements $P(i)$ is related to
$n_P^{V^H}(i)$	Number of hexahedral volume elements $P(i)$ is related to
$n_P^{S^Q}(i)$	Number of surface quadrilaterals $P(i)$ is related to
$n_P^{S^T}(i)$	Number of surface triangles $P(i)$ is related to
p	Pressure
$p_c^{start}(c)$	Address pointer to the first face of color c
$p_c^{stop}(c)$	Address pointer to the last face of color c
$\vec{q}_{1...4}$	Contributions to normal vector of a surface quadrilateral to the four corner points
r	Limiter coefficient, right state
t	Time
t	Triangle
$\vec{t}$	Normal vector of triangle t
Δt	Time step size
∇u	Gradient vector of the variable u
$\nabla^2 \vec{W}$	Laplacian of conservative variables
$\nabla^2 p$	Laplacian of pressure
$\vec{v}$	Velocity vector
vn	Normal velocity
u, v, w	Velocities in the coordinate direction

Greek symbols

Θ	Limiting factor
Θ^c	Limiting factor for the corrections
ϕ	arbitrary function, e.g. ρ, u, v, w, p
Φ	AUSM-coefficient
α	Incidence
$\alpha_{1,2}$	Angle between boundary face normal vector and vector between boundary point and field point
$\alpha_{1,2}$	Stretching coefficients
γ	Ratio of specific heats
ε_{Fan}	Area relation A_∞ / A_{Fan}
$\varepsilon^{k(2,4)}$	Coefficients to control the amount of 2nd and 4th order dissipation
θ^c	Face value of limiting factor
ν	Pressure switch controlling the second order dissipation
ρ	Density
ρ_t	Local residual
φ	Angle between point stretching vector and face normal vector

ψ	Limiter for the gradients - upwind scheme

Subscripts

Ex	Exhaust plane
F	Face
Fan	Fan inflow
g	grid level (Pre-subscript)
L, R	Left and right hand side (of the actual face)
app, lve	Approaching/leaving (the boundary face)
b	Boundary
bnd	Boundary value
$corr, est$	Correct and estimated (Area ratios)
$corr, g$	Corrected and given (velocities)
lim	Limited
na	not agglomerated
old	old, before adding of contributions
x, y, z	Derived in coordinate direction
∞	Free stream value
0	Total values

Superscripts

c	Convective
m	mirrored
n	normalized
u	Related to the variable u
x, y, z	Related to coordinate direction
Σ	Sum
$(0), (a), (b)$	Status at the begin of the time step, after the time step and after the correction
$*$	Critical values
$'$	Derivative

1 Preprocessing

1.1 Introduction

Important grid dependent information required by the solver is obtained using the preprocessing code module. Preprocessing is completed before the solver is started, since the preprocessing routines are too expensive to compute during the iterative calculation of the flow field [1]. The primary grid data (which must be generated through a third-party package) is used by the preprocessor to generate all other grid data necessary for the flow solver, for example information about the control volumes and their connectivity. In this report the algorithms used in the preprocessing module are grouped according to the type of function they provide:

- generation of derived primary grid data for use in other preprocessing functions,
- generation of secondary grid data
- generation of multilevel grids for the multigrid algorithm [2],
- assortment (coloring) of data for efficient vectorized computation,
- domain decompositioning for parallel computation [3].

1.2 The Primary Grid

The initial grid consists of polyhedral elements with triangular and quadrilateral surfaces. The quadrilateral surfaces are not always planar, however the hexahedral, prismatic and pyramidal elements will be called hexahedra, prisms and pyramids instead of more formal mathematical definitions. A fourth element type, tetrahedra, is allowed in the initial grid. The philosophy behind this choice of fundamental control volume elements can be summarized by the following:

- Tetrahedra enable local adaptation and are useful in regions where the flow field should be isotropically resolved.
- Prisms are useful in regions where two directions should be resolved isotropically, but the third direction should be resolved in a different length scale.
- Hexahedra are useful in regions where all three space directions should be resolved according to different length scales.

[1] Some grid dependent quantities, for example the norm of an area vector, are not calculated during preprocessing as the resulting memory overhead is sufficiently large to justify the addition of a small amount of extra work into the solver.

[2] see Part 2.5.7

[3] see Part 6

- Pyramids are useful to bridge between the different elements listed above.

An essential feature of the primary grid is regularity, where a nonempty intersection of two elements can be characterized as only one of the following cases, with no other type of intersection being allowed:

- Two elements may share one common corner point.
- Two elements may share a common edge, or
- two elements may share a common face.

Primary grid control volume elements are described by their corner points given in a fixed order as seen in the figures below. The orientation of points on a basic surface is initially chosen so that a curve joining the points is positive in the sense of the right-hand rule. As will be seen later, the point order is not fixed but may be changed to satisfy other criterion. The choice of the basic surface is a triangle for a tetrahedral volume element. For prism volume elements quadrilateral surfaces are chosen. Each surface of an element located inside the

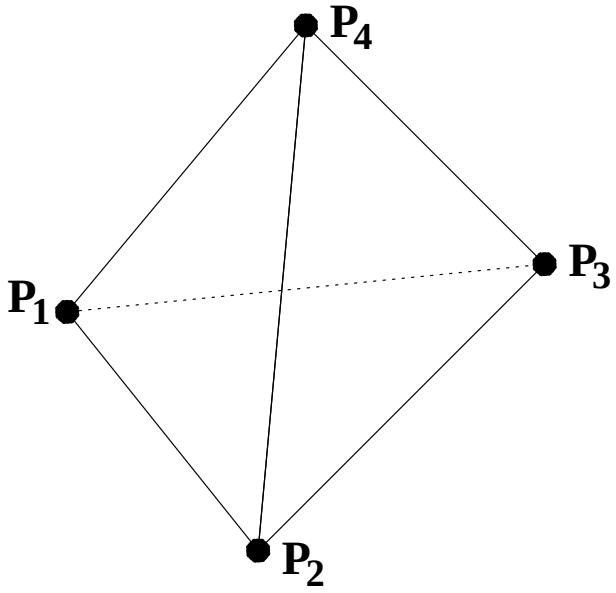


Figure 1.1: Tetrahedron$_i = \{P_1, P_2, P_3, P_4\}$

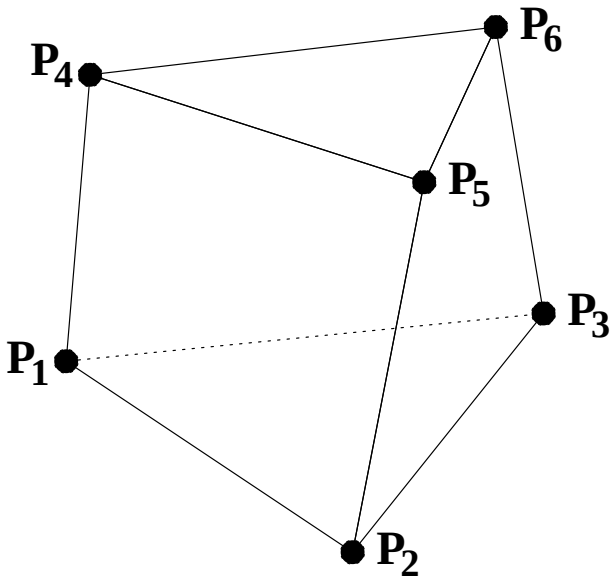


Figure 1.2: Prism$_i = \{P_1, P_2, P_3, P_4, P_5, P_6\}$

computational domain belongs to exactly two different elements, while a surface which is part of the computational domain boundary belongs to one element only. To identify these boundary surfaces, and connect them to different boundary markers, the primary grid provides triangular and quadrilateral surface elements. These elements are described by the surface corner points and a marker for the kind of boundary. Boundary markers are discussed in Part 1.3.3.

The information provided by the primary grid is summarized as follows:

- the total numbers of points and of each volume and surface element type,
- the coordinates of each point,
- the number of points belonging to each element,
- for each surface element a boundary marker to show the flow solver which boundary condition should be satisfied on this surface.

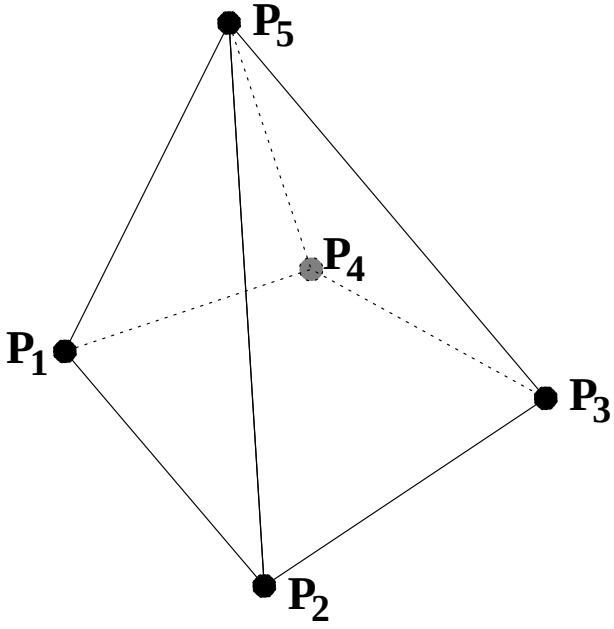


Figure 1.3: Pyramid$_i = \{P_1, P_2, P_3, P_4, P_5\}$

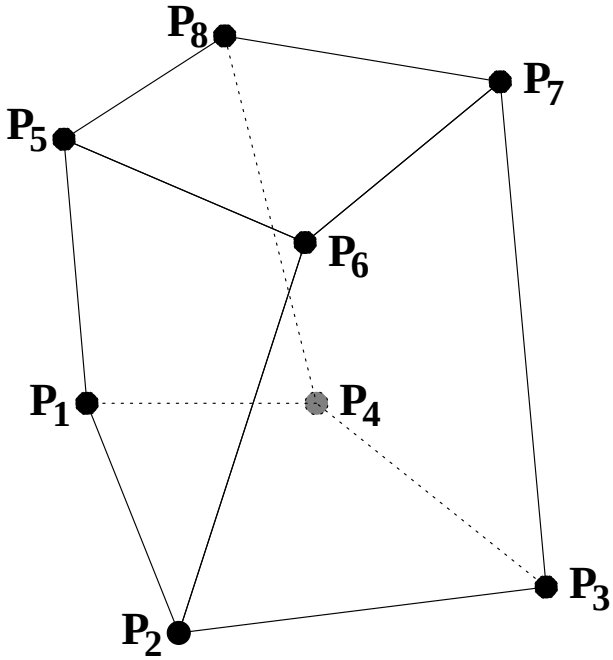


Figure 1.4: Hexahedron$_i = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$

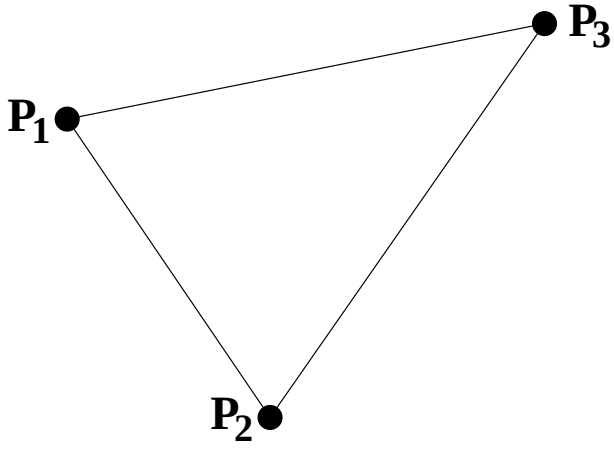


Figure 1.5: Surface triangle$_i = \{P_1, P_2, P_3\}$

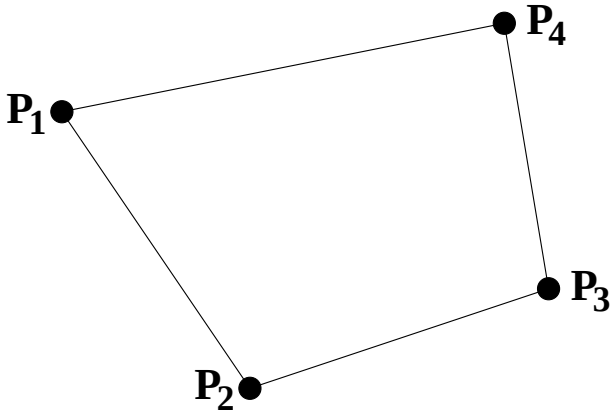


Figure 1.6: Surface quadrilateral$_i = \{P_1, P_2, P_3, P_4\}$

Each point P_i of the grid is identified by a number, which is used as a point index. The connectivity between elements is provided by their common points. The physical coordinates of a point $\vec{X}(P_i)$ are defined by the set $\vec{X}(P_i) = (x(P_i), y(P_i), z(P_i))^T$.

1.2.1 Check of the Volume Element Orientations

As the correct orientation of the volume elements cannot be guaranteed the orientation is checked and corrected if necessary. This check is done element by element. For the tetrahedron$_i = \{P_1, P_2, P_3, P_4\}$, presented Figure 1.1, the normal vector $\vec{n}$ of the triangle $\{P_1, P_2, P_3\}$ is computed by taking the following cross product:

$$\vec{n} = \frac{1}{2}\left(\vec{X}(P_2) - \vec{X}(P_1)\right) \times \left(\vec{X}(P_3) - \vec{X}(P_1)\right).$$

If the point P_4 is not located on the side of the plane of the triangle to which the computed normal vector $\vec{n}$ is pointing and the condition $\vec{n} \cdot \left(\vec{X}(P_4) - \vec{X}(P_1) \right) < 0$ is true, the orientation of the tetrahedron is reversed by changing the sequence of the corner points; for example the order of points for the tetrahedron becomes $\{P_2, P_1, P_3, P_4\}$.

A comparable correction is not possible for the other volume elements. The only further check implemented is a test for the orientation of the basic triangle of each
$\text{prism}_i = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ (see Figure 1.2). The normal vector $\vec{n}$ of the basic triangle is calculated as above and compared with the mean value of the vectors pointing from a point of the basic triangle to the related point of the other surface triangle. If the condition

$$\vec{n} \cdot \left(\vec{X}(P_4) - \vec{X}(P_1) + \vec{X}(P_5) - \vec{X}(P_2) + \vec{X}(P_6) - \vec{X}(P_3) \right) < 0.$$

is true, the order of the points defining the prism is changed to $\{P_2, P_1, P_3, P_5, P_4, P_6\}$.

1.2.2 Check of the Surface Element Orientations

The orientation of a surface element $\{P_1, P_2, P_3, ...\}$ is determined by the definition that the normal vector

$$\vec{n} = \frac{1}{2} \left(\vec{X}(P_2) - \vec{X}(P_1) \right) \times \left(\vec{X}(P_3) - \vec{X}(P_1) \right)$$

should point *outside* the computational domain. The orientation of the surface element is checked by examining the attached volume element. The volume element can be detected employing the point to volume element connectivity described in Part 1.3.1. If the normal vector points towards the interior the volume element, the first and third points forming the corners of the surface element are reversed: $\{P_1, P_2, P_3, ...\} \rightarrow \{P_3, P_2, P_1, ...\}$.

1.3 Primary Grid Data structures

The data structure of the primary grid offers the opportunity to easily loop over all volume elements. For efficient calculations structures which enable access to all points, and hence access to the volume elements or to the edges sharing the current point, are used.

1.3.1 The Point to Element Connectivity

The point to volume element connectivity is stored in two vectors. The first vector contains an index (for each point) which gives the starting location of the point element list in the second vector. The element list of a point ends at the starting entry for the list of the next point. Each different volume element gets a unique number by adding the total number of tetrahedra to each number of a prism, the sum of the total numbers of tetrahedra and prisms to each number of a hexahedra, and by an analog incrementing of the number of each pyramid. The structures may

$$index[] = \{index[P_1], index[P_2], ..., index[P_{No.of\ points}], N^{max}_{pelem}\}$$

be written as: $$pelem[] = \{1st\,elem.(P_1), ..., last\,elem.(P_1), 1st\,elem.(P_2), ..., last\,elem.(P_{No.of\ points})\}.$$

The total number of point to element relations can be expressed by the total numbers of the volume elements as:

$$\begin{aligned} N^{max}_{pelem} &= 4 * No.of\,tetrahedra + 6 * No.of\,prisms \\ &+ 8 * No.of\,hexahedra + 5 * No.of\,pyramids. \end{aligned}$$

1.3.2 The Point to Face Connectivity

The data structure of the point to face connectivity is similar to the data structure of the point to volume element connectivity described in Part 1.3.1. It consists of two vectors:

$$index[] = \{index[P_1], index[P_2], ..., index[P_{No.of\,points}], N^{max}_{pface}\}$$

$$pface[] = \{1st\ face(P_1), ..., last\ face(P_1), 1st\ face(P_2), ..., last\ face(P_{No.of\,points})\}$$

The total number of point to face relations N^{max}_{pface} is equal to twice the total number of faces.

1.3.3 Mapping of the Boundary Marker

The flow solver handles different *boundary types*; for example *symmetry plane*, *farfield boundary*, or *Euler wall* boundary types may be implemented in the solver. Since different computational simulations may require different boundary conditions (on the same grid), a mapping is used between the boundary markers of the primary grid and the boundary types. A boundary marker is a number that defines all surface elements lying in a respective region of the grid. Consider the example depicted in Figure 1.7, depicting possible boundary marker types for a generic aircraft grid. Let surface el[illegible]ve the boundary marker f_1, elements

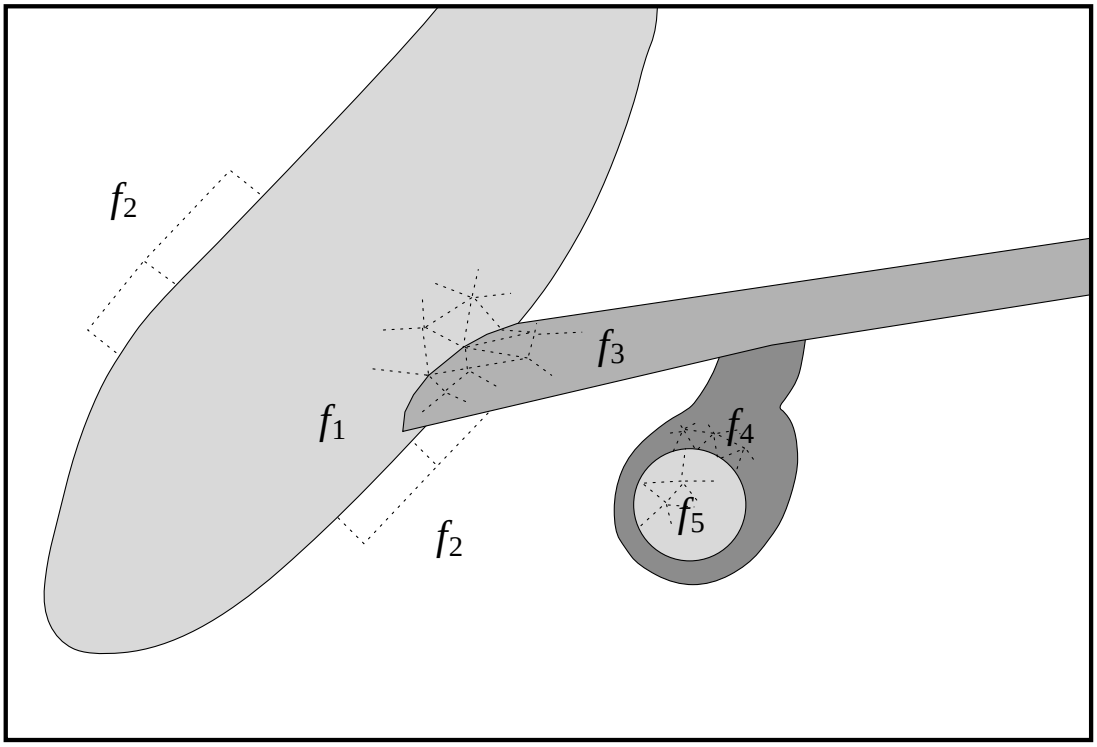


Figure 1.7: Boundary markers for a grid around an aircraft

on the symmetry plane the boundary marker f_2, elements on the wing the marker f_3, elements on the nacelle surface the marker f_4, and the elements on the nacelle inflow plane the marker f_5. The information relating boundary markers to boundary types is provided to the preprocessor by a user defined mapping file. This information is passed, together with the boundary information stored in the primary grid data, into the preprocessing algorithm. For example, all control volume faces on the symmetry plane are given a boundary marker f_2. This marker is an integer, of value 3 for example. This information is found in the primary grid data. The user must then

code a mapping file in order to associate the marker f_2 with the related boundary condition. The format and the usage of the mapping file is described in more detail in the User Guide.

Note that surface elements with different boundary markers can be of the same boundary type, e.g. if different markers are defined on a wing surface. In such a case, different markers can be assigned to a single family 'wing' which forms a *boundary part*. The boundary faces of the secondary grid (which are computed in the preprocessing step) are stored boundary part by boundary part.

1.3.4 The Edge/Face to Point Connectivity

All volume elements sharing a point P_i can be identified employing the point to element connectivity. For each element the local position of P_i can be determined and the neighboring points and edges belonging to the element are then known. To avoid double counting of edges an edge is only considered if P_j is a point with a higher index number than P_i (recall the orientation of points noted in Part 1.2), so that $P_j > P_i$. As a consequence the first point in the edge, or respectively the face, data structure is always represented by a lower point index than that for the second point.

In general each edge of the primary grid is shared by several primary grid volume elements, so that each edge $\{P_i, P_j\}$ is found more than once employing this algorithm. Hence a list L is used, initialized for all points with the integer value -1. If, during a search, a neighboring point P_j has not yet been found for P_i then $L(P_j) \neq P_i$. When a new face is stored, the list is updated so that $L(P_j) = P_i$.

1.3.5 The Boundary Face to Point Connectivity

For each boundary part one boundary face is generated around each point belonging to this boundary part. The data structure used to describe this information consists, for each boundary part, of the total number of boundary faces belonging to this part and a list of the attached boundary points. Calculating this data employing the surface element to point connectivity requires a marker to prohibit multiple counting of points.

1.4 The Secondary Grid

The secondary grid contains all grid data necessary for the flow solver. It is constructed from the primary grid data. Depending on the choice of grid metric the secondary grid differ in arrangement of control volumes and update points for the flow variables. A Cell Vertex and a Cell Centered grid metric are realized.

Note that the primary grid is required during adaptive grid generation processes and for the visualization of the data.

Cell Vertex Grid Metric

The Cell Vertex grid metric associates the flow variables with the vertices of elements of the primary mesh. Thus the primary and the secondary grid share the same points in physical space, however the secondary grid consists of control volumes surrounding each grid point. As noted in Part 1.2, the primary grid points define vertices of the primary grid control volumes. The surfaces of the primary grid control volumes are composed of the secondary grid *faces*

which are attached to each edge of the primary grid. On the boundary of the computational domain the control volumes are closed with respect to the boundary surfaces of the primary grid. The following data, which may be determined from the primary grid, is used to construct the secondary grid.

- the total number of points,
- the locations of the points in physical space,
- the sizes of the control volumes attached to each point,
- the total number of inner faces,
- the edge or face connectivity, i.e. the two points belonging to each edge,
- the size and the normal direction of the face attached to each edge,
- total numbers of boundary parts and of faces belonging to each part,
- the boundary face to point connectivity,
- the sizes and normal directions of the boundary faces.

Cell Centered Grid Metric

The Cell Centered grid metric associates the flow variables with cell centers of the control volumes of the secondary grid. The control volumes are defined by the elements of the primary grid. The cell centers are the barycenters of the control volumes. Thus the primary and the secondary grid share the same volumes in physical space.

1.4.1 Definition of the Secondary Grid Control Volumes

Cell Vertex The elements which determine the secondary grid control volume structure are described below.

- *Grid Points*
  Each grid point P_i is attached to one control volume. Hence the control volumes, and all data attached to them, may be uniquely referred to by P_i.
- *Edges*
  For each edge $\{P_1, P_2\}$ the separation point S^e between the two control volumes is given by bisection;
  $$\vec{X}_{S^e} = \frac{1}{2} \cdot \left(\vec{X}(P_1) + \vec{X}(P_2) \right).$$
- *Triangles*
  For each triangle $\{P_1, P_2, P_3\}$ the definition of the point S^t inside the triangle is sufficient to separate the control volumes by the lines $\{S^t, S_1^e\}, \{S^t, S_2^e\}$ and $\{S^t, S_3^e\}$. Defining S^t as the barycenter of the triangle guarantees that S^t is located inside the triangle.
  $$\vec{X}_{S^t} = \frac{1}{3} \cdot \left(\vec{X}(P_1) + \vec{X}(P_2) + \vec{X}(P_3) \right)$$

As seen in Figure 1.8, this configuration may result in poor quality control volumes for stretched triangles: The normal vector of the control volume boundary part $\{S^t, S_2^e\}$ becomes more normal to the edge direction $\{P_2, P_3\}$. As the control volume quality deteriorates an accompanying loss of numerical precision can be expected. Different methods of defining the secondary control volumes may be of advantage in such situations.

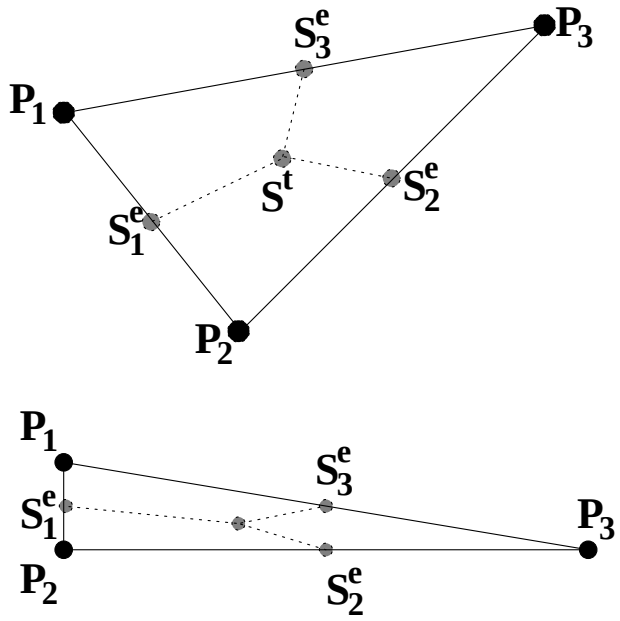


Figure 1.8: Control volume borders on a triangle

- *Quadrilaterals*
  A typical quadrilateral $\{P_1, P_2, P_3, P_4\}$ is shown in Figure 1.9. A point S^q is defined by the barycenter of the quadrilateral, given by

$$\vec{X}_{S^q} = \frac{1}{4} \cdot \left(\vec{X}(P_1) + \vec{X}(P_2) + \vec{X}(P_3) + \vec{X}(P_4) \right).$$

  Even if the quadrilateral does not lie in a plane, the barycenter is the cross point of the side bisection lines. The same problem, as noted for the case of a highly anisotropic triangle, will occur when the angles between connected edges have significant departure from orthogonality.

- *Tetrahedra*
  For each tetrahedron $\{P_1, P_2, P_3, P_4\}$, for example that shown in Figure 1.10, the point S^{tetra} inside the tetrahedron is used to subdivide the control volumes. In this example the control volumes around P_1 and P_2 have the triangles $\{S^{tetra}, S_2^t, S_1^e\}$ and $\{S^{tetra}, S_1^e, S_1^t\}$ in common. Defining S^{tetra} as the barycenter of the tetrahedron gives

$$\vec{X}_{S^{tetra}} = \frac{\left(\vec{X}(P_1) + \vec{X}(P_2) + \vec{X}(P_3) + \vec{X}(P_4) \right)}{4},$$

  and guarantees that S^{tetra} is located inside the tetrahedron and that the triangles located at one edge are coplanar. Similar problems, as in the case of a stretched triangle, will occur for an anisotropic tetrahedron.

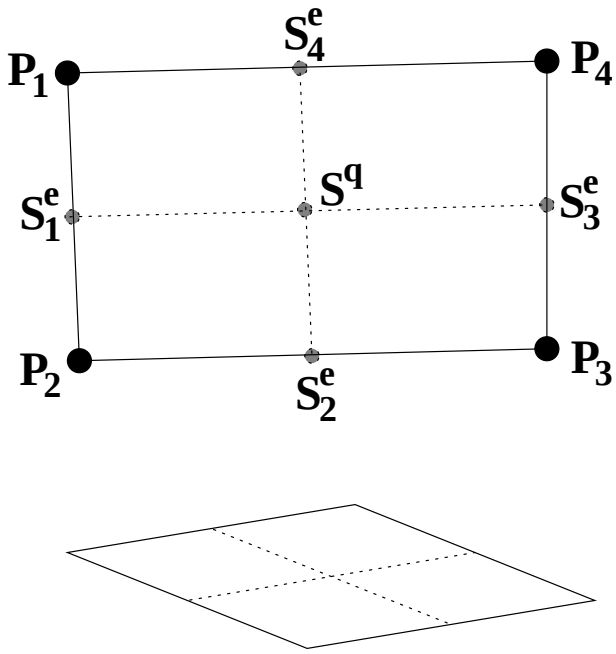


Figure 1.9: Control volume borders on a quadrilateral

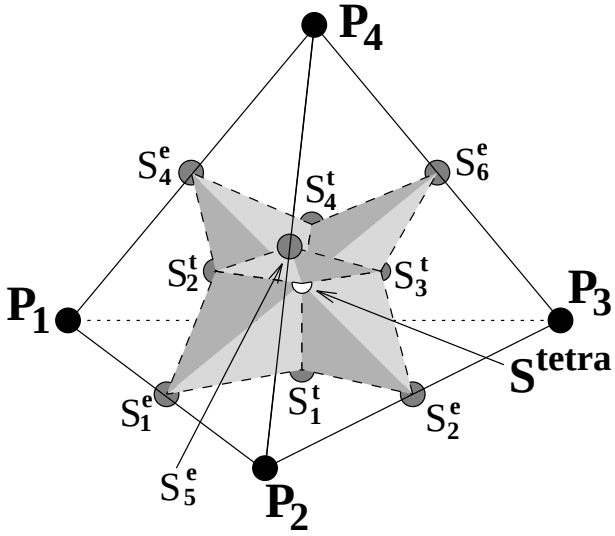


Figure 1.10: Control volume borders in a tetrahedron

- *Prisms*
  For each prism $\{P_1, P_2, P_3, P_4, P_5, P_6\}$ the definition of one point S^{prism} inside the prism is enough to separate the control volumes. The choice of the barycenter of the prism to be S^{prism} results in the following formula

$$\vec{X}_{S^{prism}} = \frac{\left(\vec{X}(P_1)+\vec{X}(P_2)+\vec{X}(P_3)+\vec{X}(P_4)+\vec{X}(P_5)+\vec{X}(P_6)\right)}{6}.$$

  If the prism is convex the barycenter lies inside the volume.

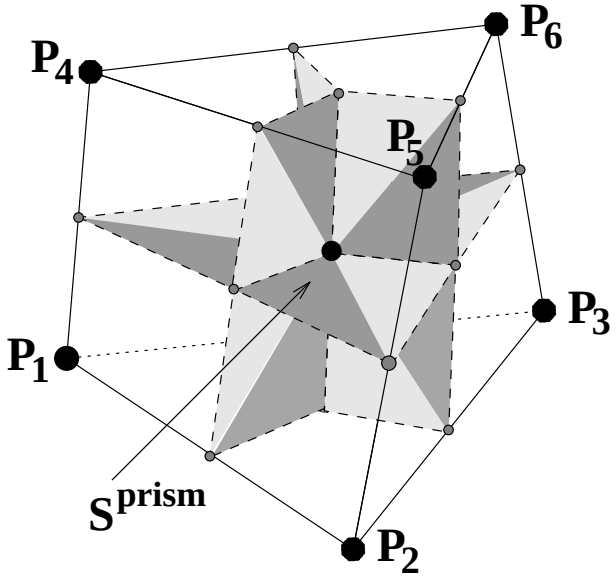


Figure 1.11: Control volume borders in a prism

- *Pyramids*
  For each pyramid $\{P_1, P_2, P_3, P_4, P_5\}$ the definition of one point S^{pyra} inside the pyramid is enough to separate the control volumes. The choice of the barycenter of the pyramid to be S^{pyra} provides the formula

$$\vec{X}_{S^{pyra}} = \frac{\left(\vec{X}(P_1) + \vec{X}(P_2) + \vec{X}(P_3) + \vec{X}(P_4) + \vec{X}(P_5)\right)}{5}.$$

  If the pyramid is convex, the barycenter lies inside the volume.

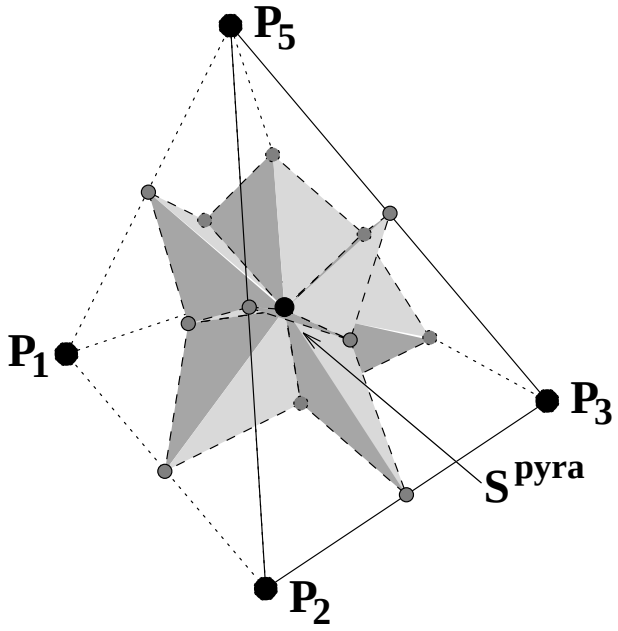


Figure 1.12: Control volume borders in a pyramid

- *Hexahedra*
  For each hexahedron $\{P_1, P_2, P_3, P_4, P_5\}$ the definition of one point S^{hexa} inside the hexahedron is enough to separate the control volumes. The choice of the barycenter of the hexahedron to be S^{hexa} gives the following formula

$$\vec{X}_{S^{hexa}} = \frac{\left(\vec{X}(P_1) + \vec{X}(P_2) + \vec{X}(P_3) + \vec{X}(P_4) + \vec{X}(P_5) + \vec{X}(P_6) + \vec{X}(P_7) + \vec{X}(P_8)\right)}{8}.$$

  If the hexahedron is convex, the barycenter lies inside the volume.

From these definitions all areas on a surface element attached to a control volume, as well as all attached surfaces inside a volume element, consist of triangular elements. Consequently the control volume consists of tetrahedra created by these elements connected with the point attached to the control volume.

Cell Centered The components which determine the secondary grid volume structure are the primary elements and their cell centers. Each element i of the primary grid defines a control volume V_i of the secondary grid. A cell center is attached to each control volume V_i. The cell center is defined as barycenter of the primary element determined by N primary grid points P_j

$$\vec{X}_i = \frac{1}{N} \sum_{j=1}^{N} \vec{P}_j.$$

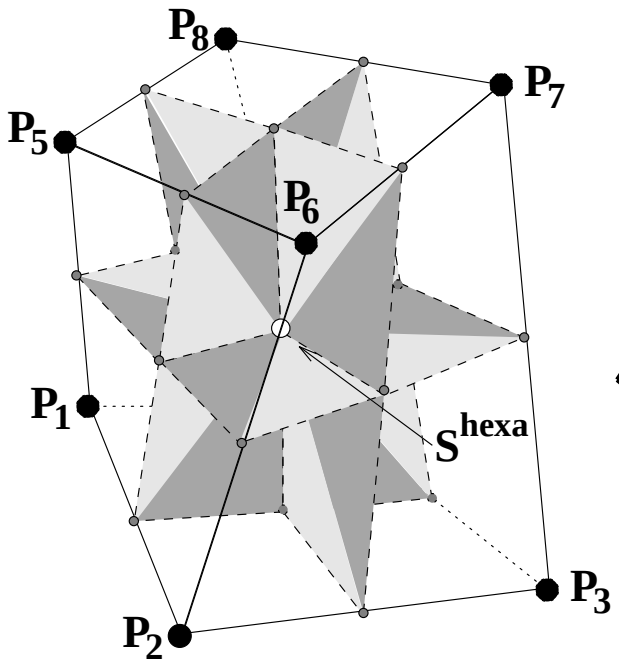


Figure 1.13: Control volume borders in a hexahedron

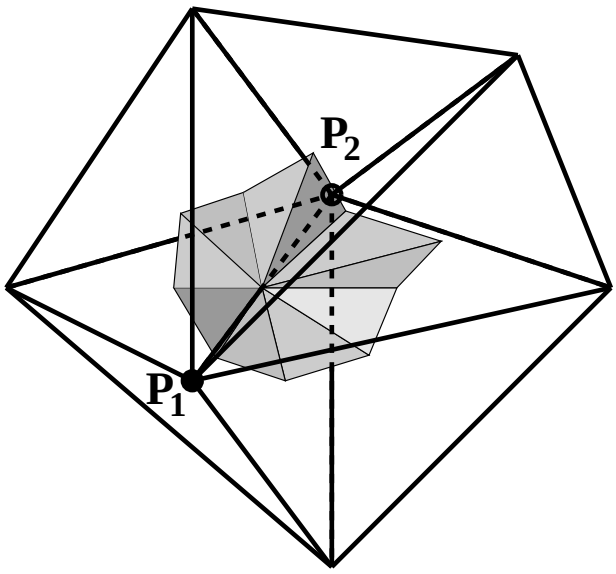


Figure 1.14: A face of the secondary grid

A cell center is called a point of the secondary grid. Thus the control volumes of the secondary grid, and all data attached to them, are uniquely referred to by the secondary grid points P_i.

1.4.2 Inner Faces

Cell Vertex Each edge of the primary grid connects two points P_1 and P_2. The attached control volumes share some triangles on their common surface according to the definitions of the secondary grid, as depicted in Figure 1.14. All these areas are summed for the face F attached to the edge$\{P_1, P_2\}$ by adding the area vectors with respect to the orientation of the edge from P_1 to P_2.

Cell Centered An inner face of the primary grid connects two elements i and j. The secondary grid points P_i and P_j of these elements define the corresponding face$\{P_i, P_j\}$ of the secondary grid. Thus an inner face$\{P_i, P_j\}$ of the secondary grid is equal to the common inner face of the primary grid elements i and j.

1.4.3 Boundary Faces

Cell Vertex All parts of surface elements belonging to one boundary type are collected together for each control volume as one boundary face. For example, Figure 1.15 shows the control volume around a point P_i which is located on the boundary of the grid. Two different boundary types come together in point P_i, represented by the light and the dark gray shading. Hence, two different boundary faces each with one normal vector are required to properly describe the control volume surrounding P_i.

Cell Centered The boundary faces of the secondary grid are equal to the boundary elements of the primary mesh. Thus the boundary types of the primary grid boundary elements are

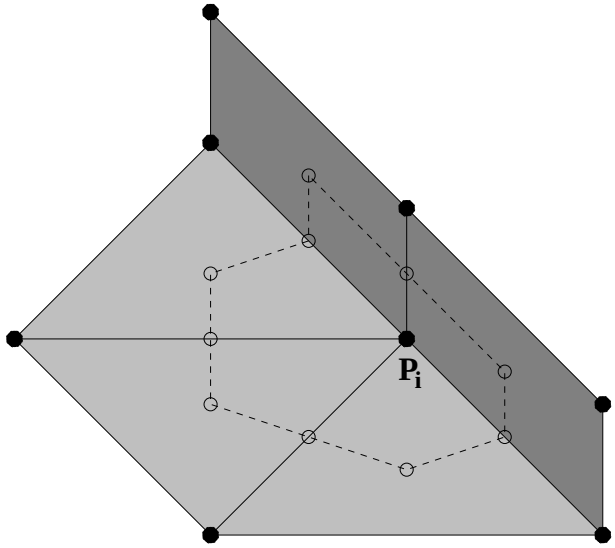


Figure 1.15: Contributions of boundary elements to the boundary faces of point P_i

assigned to the corresponding boundary faces of the secondary grid.

1.4.4 Computation of Face Normals and Control Volumes

Cell Vertex As noted above each control volume consists of some tetrahedra with triangular surface elements. The corner points of these tetrahedra and triangles are given by the secondary grid data. The area vector $\vec{n}_1$ of a given triangle $t_1 = \{\vec{X}_1, \vec{X}_2, \vec{X}_3\}$ can be written as [4]:

$$\vec{n}_1 = \frac{1}{2}\left(\vec{X}_2 - \vec{X}_1\right) \times \left(\vec{X}_3 - \vec{X}_1\right).$$

Normally each triangle t_1 occurs together with a second triangle $t_2 = \{\vec{X}_3, \vec{X}_2, \vec{X}_4\}$. The vectorial sum of the area vectors for both triangles is given by:

$$\vec{n}_1 + \vec{n}_2 = \frac{1}{2}\left(\vec{X}_4 - \vec{X}_1\right) \times \left(\vec{X}_3 - \vec{X}_2\right).$$

The volume V_1 of a tetrahedron $\{\vec{X}_1, \vec{X}_2, \vec{X}_3, \vec{X}_4\}$ is given by:

$$V_1 = \frac{1}{6}\left(\vec{X}_4 - \vec{X}_1\right) \cdot \left(\vec{X}_2 - \vec{X}_1\right) \times \left(\vec{X}_3 - \vec{X}_1\right).$$

The number of operations required to calculate the total volume can be reduced if the neighbored tetrahedron $\{\vec{X}_1, \vec{X}_3, \vec{X}_5, \vec{X}_4\}$ is taken into account:

$$V_1 + V_2 = \frac{1}{6}\left(\vec{X}_4 - \vec{X}_1\right) \cdot \left(\vec{X}_2 - \vec{X}_5\right) \times \left(\vec{X}_3 - \vec{X}_1\right).$$

The sign of the calculated volume should be positive, if

- the orientations of the tetrahedra are correct in the sense of Part 1.2.1 and
- the involved elements of the primary grid are convex enough, so the definitions of the control volumes provide points located inside the respective element.

[4] The sign can be chosen according to the defined direction of the face

Cell Centered The computation of face normals and of volumes of control volumes follows the same principles as the Cell Vertex Computation.

1.5 Minor Additional information

1.5.1 Determination of the Neighboring Point

For some boundary types it is essential to involve information coming from the flow field. Therefore it is necessary to determine, for each boundary point, one point that is located normal to the boundary. The alignment of a boundary face related to a point P_i is described by its normal vector $\vec{n}_f$. Employing the point to face connectivity it is possible to identify all faces P_i is related to and by that all neighboring points. For each neighboring point P_j the cosine of the angle α_n between the face normal and edge $\{P_i, P_j\}$ is given by:

$$\cos\alpha_n = \frac{\left(\vec{X}(P_j) - \vec{X}(P_i)\right) \cdot \vec{n}_f}{\left|X(P_j) - \vec{X}(P_i)\right| \cdot \left|\vec{n}_f\right|}.$$

The point that is located most normal to the boundary can be identified by the smallest value of $\cos\alpha$. In Figure 1.16 the determination of the reference point for a boundary point P_i is

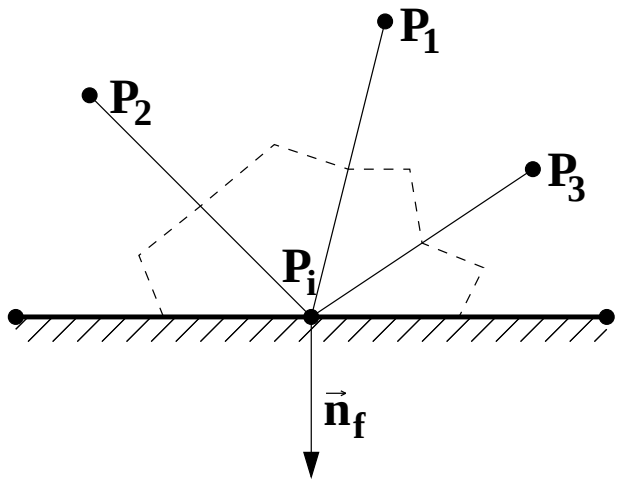


Figure 1.16: Neighboring points of a boundary point P_i

illustrated. The point P_1 is the point that is located most normal to the boundary.

1.5.2 Computation of the Fan Area

In order to compute the flow through a nacelle correctly, the area of the fan inflow face and the direction of the nacelle axis are required. The nacelle axis is assumed to be normal to the fan inflow face. The vectorial sum of all area vectors of the boundary faces belonging to boundary part *nacelle inflow* provides the area of the fan inflow face. Note that if the configuration has more than one nacelle the axes are assumed to be parallel and the area relations are assumed to be identical.

1.6 Vectorization Concerns

The scheme described here is based on the faces of the secondary grid. Therefore, a major part of the calculations in the flow solver is performed by considering the grid faces one by one, while the computed values (e.g. fluxes over the face) are adjoined to the respective points on both sides of the face. The architecture of modern vector computers is designed such that l_{vec} operations can be done at the same time, with l_{vec} being the *vector length* of the respective computer. So the fluxes over l_{vec} faces can be computed and adjoined to the related points at the same time. If one grid point is related to more than one of the l_{vec} faces, the association of the computed values cannot be done correctly. Hence, coloring of the inner faces is necessary to exploit the capabilities of modern vector computers by avoiding such conditions. Only faces of identical color are allowed to be considered at the same time.

1.6.1 Coloring of the Inner Faces

The coloring is performed by considering all faces $\{F_{start},\ldots F_{No.of\ faces}\}$ of the grid that have not been colored yet. The variable F_{start} is pointing to the first face that is not colored. Before the coloring, F_{start} is pointing to the first grid face. Whenever a new face is colored, F_{start} is incremented . The coloring process is finished when F_{start} equals the total number of faces. When no face is found that can be colored with the current color c, but not all faces are colored, the color c is incremented and the remaining faces are again examined.

$$\{p_c^{start}(m)\,|\,m=1,\ldots N^c\}$$

and

$$\{p_c^{stop}(m)\,|\,m=1,\ldots N^c\}$$

are generated with N^c being the total number of colors. $p_c^{start}(c)$ is pointing to the first face of the color c. Whenever c has to be incremented, the current value of F_{start} is assigned to $p_c^{start}(c)$. When all faces are examined and the color is to be incremented again, the current value of F_{start} is assigned to $p_c^{stop}(c)$. Hence, $p_c^{stop}(c)$ is pointing `behind` the last face of the color c.

As the purpose of the coloring is to avoid equal colors on faces the same point is related to, for each new face $F_i=\{P_1,P_2\}$ the end points of the face are examined to see if one of them is related to a face of the current color c. This is checked by employing a helping list $\{L_P^c(m)\,|\,m=1,\ldots No.of\ points\}$. This list is initialized with -1 for each element. If $L_P^c(P_j)=c$ then a face point P_j is colored with c. Hence, if $L_P^c(P_1)=c$ or $L_P^c(P_2)=c$, the current face F_i is not suited to be colored with c and the next face F_{i+1} is examined. If for face F_i it is found that none of the related points is related to another face of the color c, F_i is colored with c. In this case, c is assigned to $L_P^c(P_1)$ and $L_P^c(P_2)$. If one of these points is examined again, the entry in L_P^c indicates that the respective face is not suited to be colored with c. Then the number of the face F_i and all data related to F_i is exchanged with the first face in the list of faces that has currently not been reordered. The index of this face is currently stored in F_{start}. After the data exchange, F_{start} is incremented and the search continues.

When the examination is finished for the color c, the number of faces $n_c^F(c)$ of the color c can be computed as:

$$n_c^F(c)=p_c^{stop}(c)-p_c^{start}(c).$$

For performance reasons the number of faces should be divisible by the vector length of the computer l_{vec}, and $n_c^F(c)$ is chosen to be the next smaller multiple of l_{vec} (when $n_c^F(c)>l_{vec}$) and $p_c^{stop}(c)$ and F_{start} are changed respectively.

After the coloring has been performed, the faces of each color c are ordered employing a recursive ordering algorithm. Ordering criterium is the index of the first point P_1 of each face $\{P_1, P_2\}$ (note that $P_1 < P_2$, see Part 1.3.2). The sorting gives an improved memory distribution. The recursive sorting is described in the next Part 1.6.2.

1.6.2 Recursive Ordering

A recursive ordering is required at several times in the scheme. The employed algorithm is described in this section.

A vector $\{A(m) \,|\, m = a_o, \ldots b_o\}$ is to be ordered based on a certain criterium $\{B(m) \,|\, m = a_o, \ldots b_o\}$ such that

$$B(i-1) \leq B(i) \leq B(i+1) \;\text{ for }\; i = a_o + 1, \ldots b_o - 1.$$

The vector A has to be considered as a *parent part* of the level $l = 1$. Firstly, the entire vector A is split into two *child parts* of the level $l = 2$ with desirably the same size: $\{A_1(m) \,|\, m = a_{n1}, \ldots b_{n1}\}$ and $\{A_2(m), | \, i = a_{n2}, \ldots b_{n2}\}$ with

$$a_{n1} = a_o \;\text{ and }\; b_{n1} = int(0.5 \cdot b_o)$$

and

$$a_{n2} = b_{n1} + 1 \;\text{ and }\; b_{n2} = b_o.$$

Two child parts with the same parent are called *sister parts*. These child parts are ordered separately employing the algorithm described here (recursive approach), which means that the splitting continues with increasing levels until less than three elements of A are to be considered in all child parts of the level $l = l_{max}$.

$$b_{n1} - a_{n1} < 2 \;\text{ and }\; b_{n2} - a_{n2} < 2.$$

These elements are now ordered in the vectors A and B based on the entries in B, such that:

$$B(a_{n1}) \leq B(b_{n1}) \;\text{ and }\; B(a_{n2}) \leq B(b_{n2}).$$

After both sister parts are ordered, they are merged together (considering B). The parent part $\{A(m) \,|\, m = a_o, \ldots b_o\}$ of the next lower level $l = l_{max} - 1$ is obtained in the right order. The obtained parent part is now to be considered as a child part and is merged together with its (also ordered) sister part to obtain the parent part of the next lower level $l = l_{max} - 2$. When the level $l = 1$ is reached again, the vector A is ordered as desired.

Note that in order to sort a vector with N elements, $int(\log(N)/\log(2))$ levels will be created. The number of children parts increases by a factor of 2 with each level.

Care has to be taken if any data is adjoined to the elements of A. As the indices of the elements may change, the adjoined data has to be changed as well.

1.7 Multigrid Concerns

The multigrid algorithm used in the code is based on an upward cycle, hence control volumes are fused together to form a coarser grid. The secondary grid , described in the Part 1.4, forms the fine grid from which the agglomeration procedure is commenced. The coarser grid contains all information for the flow solver in a data structure similar to the data structure of the secondary grid. In addition the connectivity between the control volumes respectively the points of both grids is provided by the following structures:

- For each control volume of the fine grid the control volume of the coarse grid to which the fine grid cell is fused (the coarse grid control volume is referred to as the *parent*) is stored in a vector.
- For each control volume of the coarse grid the subset of fine grid control volumes, called *children*, of which the coarse grid cell consists, are stored in two vectors. One vector contains all children of the coarse grid, the other vector provides the starting address of the child subset belonging to each parent.

The coarse grid may be coarsened again, using the same methods described above. The agglomeration of the control volumes is performed until the desired coarsening level is achieved. To avoid poor multigrid performance the fusing procedure must preserve some topological aspects of the finest grid:

- Depending on the implementations in the flow solver different boundary conditions must not occur for the same control volume.
- The use of prisms and hexahedra in a primary grid normally introduces directions with different spatial resolutions. The desired decoupling of these flow directions present in the finest grid should not be disturbed on the coarser level.
- When the flow field around a body is computed, the farfield discretization can be quite coarse on the initial grid. Nevertheless the flow of information from farfield cells should be controlled so that the information between upstream and downstream control volumes is still passed in a physically consistent manner during the upward part of the multigrid cycle.

The multigrid procedure can be regarded to be composed of two parts. In the upward cycle, or *topological fusing part*, the parent of each fine grid control volume is determined. In the downward cycle the children of each parent are determined. In the *physical fusing part* all the secondary grid information, like location, size and surfaces of the control volumes, are calculated by using the fine grid information and the information known about the children. The coarse grids have to be checked as to whether the total size and the integrals of the surfaces of each control volume are correct. The sum of all control volume sizes has to remain constant. This check is done by adding up all control volume sizes. Furthermore, the boundaries of the control volumes have to be closed. In a loop over all faces of the current secondary grid the components of the normal vector related to the current face are added to the surface integrals of the two respective endpoints. This is also done for the boundary faces. Then the maximal integral of the surface boundary is determined and printed out for the inner points and the boundary points.

1.7.1 Topological Fusing Part

The topological fusing is performed using an advancing front method. It consists of a queuing system to find the next fine grid control volume, called seed volume, on the front and some routines to determine the subset of neighbored control volumes which will be fused with the seed volume. Each fused coarse grid control volume is attached to the total number of all coarse grid control volumes previously fused.

In order to topologically characterize the different secondary grid parts arising from tetrahedral, prismatic or hexahedral parts of the primary grid the following definitions are useful:

- Two control volumes sharing a common face are called *direct neighbors* of each other.
- A control volume P_1 is called an *indirect neighbor* of a control volume P_2 if they are not direct neighbors but have a direct neighbor in common.
- All control volumes sharing at least one point with a control volume P_1 are called the *neighborhood* of P_1.
- A *single connected neighbor* is a direct neighbor of a control volume P_1, which does not share any direct neighbors with P_1.

Based on these definitions the parts of the secondary grid can be described with:

- *Tetrahedral connectivity:*
  The connectivity of a control volume is called tetrahedral connectivity if no indirect neighbor is part of the neighborhood. In this case the number of single connected neighbors equals zero.
- *Prismatic connectivity:*
  A neighborhood of a control volume with a prismatic connectivity consists only of some direct and some indirect neighbors. Away from the boundaries there are exactly two single connected neighbors in this case. If the prisms of the primary grid include boundary surface triangles the number of single connected neighbors reduces to one. In the normally avoided case of prisms with boundary surface quadrilaterals the number of single connected neighbors can increase at configurations like a sharp trailing edge of a wing.
- *Hexahedral connectivity:*
  A control volume with a hexahedral connectivity possesses a neighborhood containing direct neighbors, indirect neighbors and indirect neighbors of direct neighbors. Being connected via hexahedra in all directions a control volume has exactly six direct neighbors all single connected. The number of single connected neighbors varies in cases of boundary contacts from at least three to more than six. If the control volume is located on the border between a hexahedral and for example a tetrahedral part of the grid (with some pyramids in between) the number of single connected points can decrease to zero.

During the topological fusing the connectivity of each control volume is described by the number of single connected neighbors. Ignoring some special cases a tetrahedral connectivity is assumed if the number is zero and a prismatic connectivity is assumed if the number is one or two. A higher number is assumed to indicate a hexahedral connectivity.

The Queuing System

The advancing front should start on special boundary parts, for example on fixed walls but not on the farfield. Within the set of volumes with the same boundary part priority, the volumes should be preferred which are located on an edge or in a corner of the computational domain. The determination of these priorities is described in Part 1.7.1.

The selection of the next seed volume within each priority set is controlled by subsets, called queues, in which they are grouped according to criterion described in Part 1.7.1 and Part 1.7.1. The search for the next seed volume starts in the queue with the lowest queue number. Within each queue the volume, which has been touched by the front first, is elected to be the next seed volume.

Volume Priorities All volumes connected to walls of a body, to inlet or outlet planes of an engine or to a symmetry axis are given a boundary part priority ($bpp = 1$). All other boundary volumes as well as all inner volumes have no special boundary part priority ($bpp = 0$).[5]

In a second step some geometric aspects separate between the volumes of the same boundary part priority ($bpp > 0$). Those volumes belonging to different planes like edge or corner volumes get a higher priority. To determine the number of planes $n_{\text{planes}}(V_i)$ a volume V_i is connected to, knowledge of the primary grid is needed, because all information about the surfaces belonging to the same boundary part are summarized in a single normal vector of the secondary grid. Therefore the volume priorities can only be calculated on the mesh of the finest multigrid level and are handed on from children to parents during the determination of the coarser, secondary grids of the higher multigrid levels. Two surface elements of a volume are thought to represent different planes if the angle between their normal vectors is greater than 45°.

The volume priority $Pr(V_i)$ is defined as:

$$\begin{aligned} Pr(V_i) &= n_{\text{planes}}^{\max} * (bpp(V_i) - 1) + n_{\text{planes}}(V_i) - 1 && bpp(V_i) > 0 \\ Pr(V_i) &= 0 && bpp(V_i) = 0 \end{aligned}$$

with the maximal occurring number of planes $n_{\text{planes}}^{\max}$ attached to one volume.

Initialization of the Queues Initially the volumes of each priority are sorted into different queues according to their connectivity to agglomerate hexahedral parts of the mesh previous to prismatic parts and at last the tetrahedral parts. The higher the number of single connected neighbors of a given volume is, the lower is the number of the queue the volume is attached to. The current implementation uses eight queues from which the lowest four can only be reached by volumes lying at the advancing front, see Part 1.7.1. In addition the queue, to which a volume belongs, is stored for every volume because a rearrangement of the queues during the update procedure is computationally too expensive.

Update of the Queues After each agglomeration step all agglomerated volumes are removed from the queues by setting the queue number of the volume to be out of the range of allowed queue numbers. The first time a volume is hit by the fusing front the number of the coarse grid volume is stored. Each time a volume V_i is touched by the front via a new coarse volume agglomerating a direct neighbor of i, the queue number of i is decremented by one till zero and V_i is added to the queue respectively. Therefore the volume V_i gets closer to become the next seed volume. Between several volumes neighbored to the same coarse grid volume the neighbors of the seed volume will be queued previous to the neighbors of the direct neighbors of the seed volume and at last the neighbors of indirect neighbors of the seed volume.

As an example the queue numbers of some hexahedra in front of the agglomeration front are depicted in Figure 1.17. The queue number of a hexahedron *qnh* away from the front is provided by the initialization, Part 1.7.1. As a result of the queuing the advancing front tries to grow in closed layers from the initially touched boundary at least in hybrid grids.

Maximal Set of Fusible Volumes

In principal a seed volume is allowed to be agglomerated with its whole neighborhood. That means, for a seed volume in a tetrahedral part of the grid, that every direct neighbor - given by the point to face connectivity - is a candidate for the maximal set of fusible volumes. In

[5] To date no advantage in this approach has been observed.

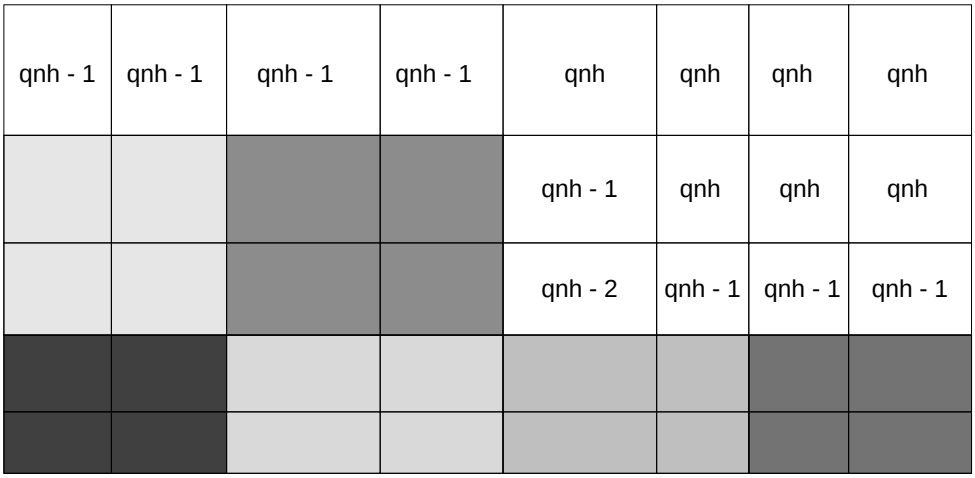


Figure 1.17: Queuing of hexahedra near the agglomeration front.

a prismatic part indirect neighbors, who have only an edge in common with the seed volume, have to be added, see Figure 1.18. They can be described as direct neighbors of the single connected neighbors which have another direct neighbor other than the seed volume or the single connected neighbor in common with the seed volume.

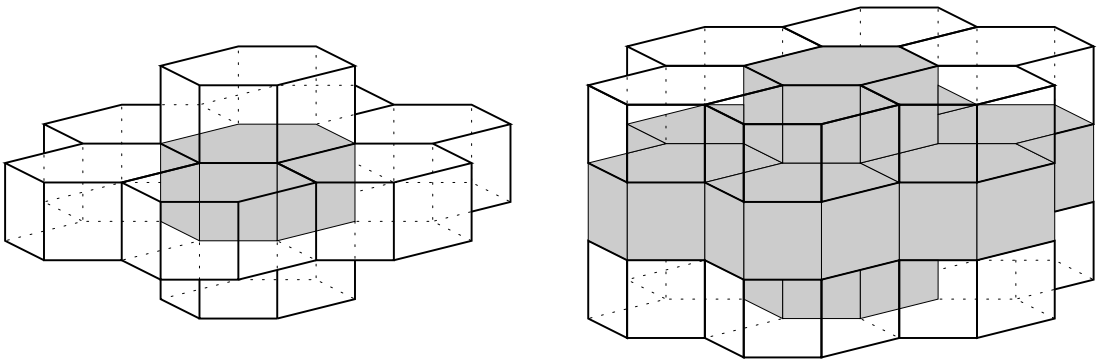


Figure 1.18: Prismatic neighborhood: seed volume and direct neighbors (left), direct and indirect neighbors (right)

In a hexahedral part additional neighbors, who only share one point with the seed volume, have to be found. They are direct neighbors of more than one indirect neighbor, who have a common edge with the seed volume. The hexahedral situation is depicted in Figure 1.19.

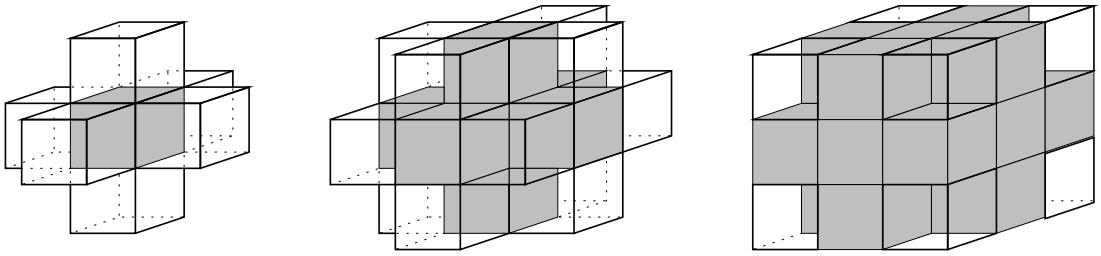


Figure 1.19: Hexahedral neighborhood: direct neighbors sharing faces (left), indirect neighbors sharing edges (middle) and neighbors sharing points (right) with the seed volume

Restrictions to the Fusible Set of Volumes

A typical coarsening rate in a structured multigrid code is 1:8 for three dimensional applications. The maximal set of volumes for this case shown in Figure 1.19 offers a coarsening rate of 1:27. Therefore it is sufficient to use only a subset of the maximal set during the agglomeration to provide good coarser grids. The most obvious restriction to the fusible set is that volumes can not be agglomerated to more than one seed volume. While the agglomeration front advances, the next seed volume is found in the neighborhood of an already fused volume.

Consider a simple structured grid over a flat plate. The algorithm starts in every corner of the plate and finds 7 neighbors for every seed point. Running along the edges of the plate the situation for the seed volume will always be the same, because the boundaries of the computational domain create again, together with the already fused volumes, a corner situation. If the number of volumes along the edges of the plate is a multiplier of 2, the complete boundary of the plate is coarsed with a 1:8 rate. Being touched by two fused neighbors the next seed volumes on the plate will be fused in the same way. After fusing all volumes on the flat plate the agglomeration front will start the next layer again in the corners, because the underlying coarse grid volumes have the lowest numbers. As long as the grid is coarsened in this structured manner, this algorithm will provide an equivalent grid.

Incompatible Boundaries Depending on the treatment of a boundary it can be important that two different boundary types do not occur at the same volume. To preserve this kind of topological information during the agglomeration procedure, the boundary types of the initial grid are investigated. Only if a volume exists in the initial mesh, which is attached to two boundary types, is the agglomeration procedure is allowed to fuse volumes of these boundary types together. Therefore, during picking up the neighbors of the seed volume to get the maximal fusible set, the compatibility of the boundary types between the already found neighborhood (including the seed volume) and the next neighbor is checked.

Cutting the Structure of an Underlying Layer In a prismatic or hexahedral region of the grid the algorithm tries to grow layer by layer over the boundary surface. A new coarse grid volume is only allowed to be on top of more than one coarse grid volume of the underlying layer if this underlying or old volume is covered completely by the new volume.

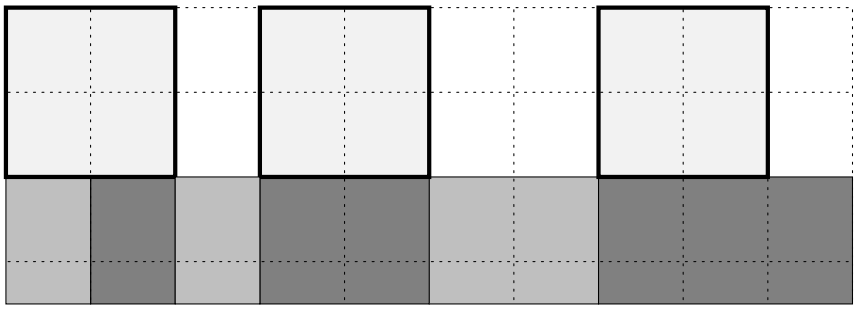


Figure 1.20: Allowed fusing: new volume covers old volumes completely (left), new volume covers exactly one old volume (middle) and new volume lies inside the cover of one old volume

The allowed fusing, simplified in two dimensions, are depicted in Figure 1.20 and a typical cut situation in Figure 1.21.

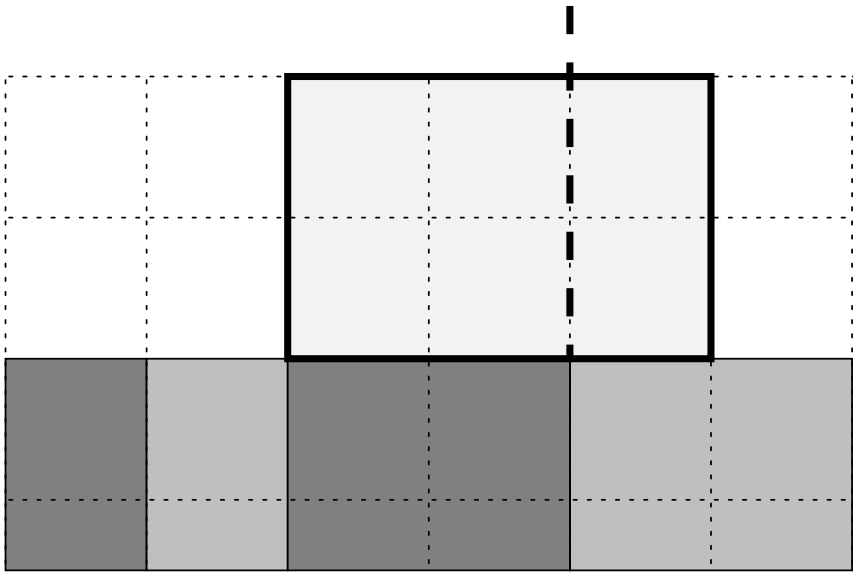


Figure 1.21: The parts of the new volume divided by the broken line are not allowed to be fused together due to the structure of the underlying layer

Fusing to Coarse Grid Volume In addition to the cutting described above, and to prohibit hardly fused volumes, an agglomeration with an already fused neighbor is allowed in special situations.[6] One situation occurs when different parts of the agglomeration front reach each other and some seed volumes are left without or with only one fusible neighbor. The other case, shown in Figure 1.22, is a result of a cutting to preserve the structure of an underlying layer.

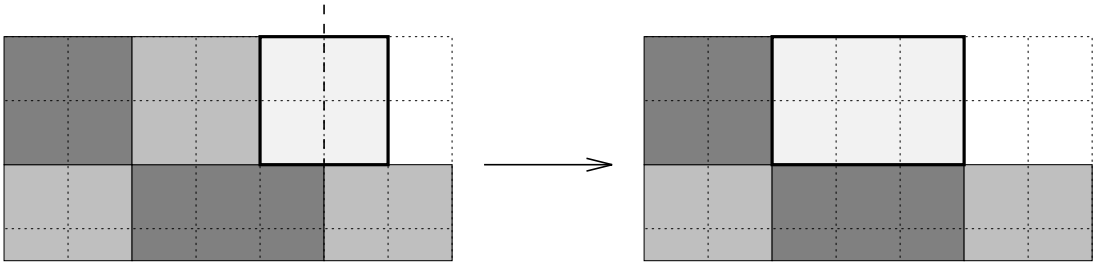


Figure 1.22: After cutting the new fusible set, some volumes are agglomerated to a previously fused cell.

Semi Coarsening In highly stretched parts of a hybrid grid it is possible to achieve smaller aspect ratios for the coarser volumes than on the initial grid. This is done be selecting only those volumes, which are connected to the seed volume via faces larger than a given fraction of the largest surface area of the seed volume, for fusing. The disadvantages of semi coarsening in this way are the small coarsening ratio (more volumes to calculate) and the corruption of the initial grid structure. Whether convergence is improved is currently not clear.

Unstructured Grid Improvement In a tetrahedral part of a grid a subset of fusible volumes can be chosen to minimize the ratio of the surface to the volume of a coarse grid cell. To become

[6] This fusing can be the origin of coarse grid cells consisting of more volumes than the complete neighborhood of any attached volume.

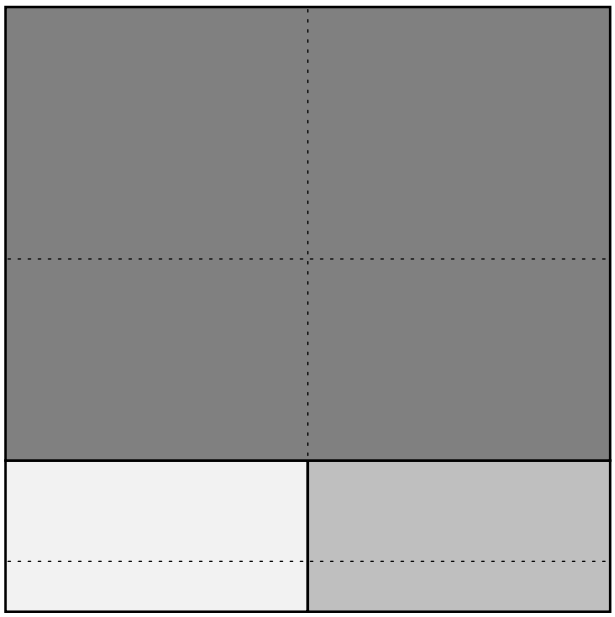


Figure 1.23: Semi coarsening with a fraction factor of 0.5

independent from the scaling units the minimizing function f is given by:

$$f = \frac{\text{surface}^{1.5}}{\text{volume}}.$$

Starting from the seed volume the algorithm searches for the next volume which provides the smallest value of f when being fused to the already determined subset. This loop terminates with a subset S_n having an optimal value f_n for each possible number n of neighbors out of the neighborhood of the seed volume. The choice of which subset will be fused is influenced by a parameter called the point fusing reward $pfr \in [1.0, \infty]$ according to the expression:

$$\min_n \left(\frac{f_n}{pfr^n} \right).$$

The sense of the pfr parameter is to decrease the influence of f and allow more volumes to be fused. In practice values of pfr between 1.0 and 1.2 result in semi-coarsening behavior, while values above 1.6 result in minimal changes compared with an agglomeration for which no selection according to a penalty function has been performed.

Limiting the coarse grid volume size To avoid grids which are too coarse to get any multigrid improvements, the maximal size of a fusible volume is restricted to a fraction of the size of the complete computational domain. This fraction is chosen as $\frac{1}{256}$, which stops the agglomeration on the coarsest level for typical three dimensional cases with about five volumes in every direction.

1.7.2 Fusing Part

The fusing part includes:

- determination of the physical coordinates of the coarse grid control volumes and their sizes,

- determination of the relationship between the volumes of the coarse grid and the fine grid,
- calculation the inner faces,
- computation of the boundary faces.

Physical Coordinates and Control Volume Sizes

From the topological fusing part the parent volume in the coarse grid are known for each fine grid volume. Hence, within a loop over all fine grid volumes the sizes of the volumes with the same parent can be added up. The result yields the sizes of the control volumes of the coarse grid. The physical coordinates of the points attached to the coarse grid volumes can be determined in two different ways, which are shown in Figure 1.24. In the first case, the coordinates of a coarse grid volume is set equal to the coordinates of the seed volume in the fine grid. In the second case, the volume weighted average of all children volumes is used. The second approach can provide some advantage for the interpolation of the corrections, but only in the first case will the coordinates belong always to a point inside the control volume. Nevertheless the influence of the choice is small due to first order approximations made on the coarser multigrid levels.

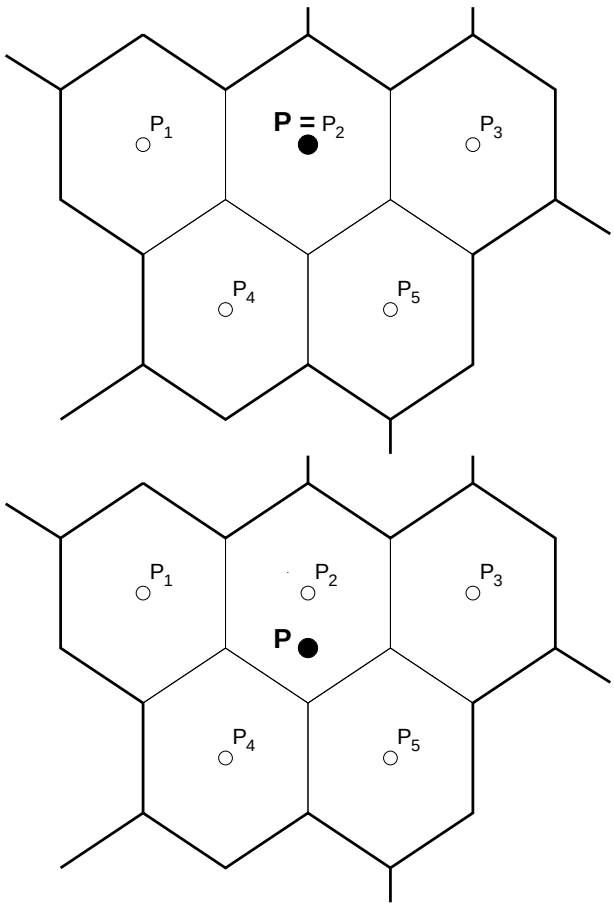


Figure 1.24: Coarse grid coordinates remaining in seed volume position and shifted to averaged position

Relationship between Coarse Grid and Fine Grid Volumes

The information about the parent of every fine grid volume is stored in a vector. The opposite information about the children belonging to each coarse grid volume needs a data structure similar to the structure described in Part 1.3.1.

$$index[] = \{index[V_1], index[V_2], ..., index[V_{No.of\,coarse\,grid\,vol.}], No.of\ fine\,grid\,vol.\}$$

$$child[] = \{1st\,child(V_1), ..., last\,child(V_1), 1st\,child(V_2), ..., last\,child(V_{No.of\,coarse\,grid\,vol.})\}.$$

Computation of the Faces

If a child of volume V_i has a face to a child of volume V_j, then V_i and V_j have a common face. If two volumes are connected by more than one face between their children, the normal vectors of the fine grid are added up with respect to the orientation of the face on the coarse grid. A typical situation is depicted in Figure 1.25. The fine grid boundary faces belonging to the children of

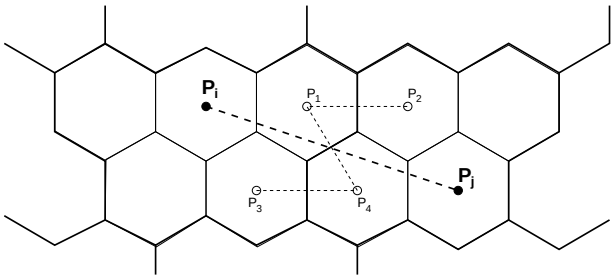


Figure 1.25: Determination of coarse grid face vectors

one coarse grid volume are also agglomerated separately for each boundary part. Identically to the interior faces, the normal vector of a coarse boundary face can be calculated by adding the normal vectors of the respective fine grid faces. The orientation of the normal vectors does not have to be considered, since these vectors always point outside the domain.

1.8 Chimera method

1.8.1 Introduction

The overset grids method is originally introduced to deal with a complicated configuration with structured meshes. The method subdivides the computational domain into simpler subdomains which admit a more easily constructed mesh. The technique do not require common boundaries between subdomains, but rather, a common or overlap region is required to provide the means of matching the solutions across boundary interfaces. The usual procedure uses interpolation of embedded boundaries to provide the necessary communication among the computational meshes and hole-cutting of the mesh becomes out of the computational region. Steger et al. [68], Benek et al. [7] developed overset grids method in two and three dimensions for Euler Equations. Dougherty [18] and Dougherty et al. [19] extended this technique to allow movement of embedded meshes. This method is quite powerful for dealing with the moving bodies because remeshing is not required. However, this method also has several shortcoming; such

as a loss of conservation, extra cost due to interpolation and locally reduced accuracy due to mismatched cell sizes between the computational meshes.

To apply this method to the unstructured mesh, three major steps are needed:

1. The creation of the overlapping region, this step is part of the grid generator, the user should specify the analytic shapes, such as cubes,spheres, cones, etc. as hole cutting geometries.Automatic approach for unstructured grid was introduced by Nakahashi [55].

2. The cell surrounding each interpolation point is searched from the other grids, using Alternating Digital Tree ADT [8].

3. The conservative variables are transferred by linear or trilinear interpolation [42].

Unsteady prediction of viscous flows with relative movement between component parts remains an extremely challenging problem facing CFD. There are a number of difficult problems for which dynamic overset grid schemes have been exclusively applied, including space shuttle booster separation [49], helicopter with moving rotor blades [50], and manoeuvring aircraft by CFD aerodynamic and flight mechanic coupling [42]. Navier-Stokes calculations with these methods require high computational resources. Parallel computing offers a very effective way to improve the productivity in doing CFD. Therefore the purpose of this study is to develop an efficient parallel computation algorithm for analyzing the flowfield of complex geometries using overset unstructured grids technique [43, 44].

1.8.2 Stencil search

To exchange the information between the grids, an efficient and reliable search algorithm must be developed. Alternating Digital Tree is efficiently utilized in the present method. The steps involved in generating an ADT and searching are outlined as follows:

1^{st}step: Determine the bounding boxes $(\{x_{min}, y_{min}, z_{min}\}, \{x_{max}, y_{max}, z_{max}\})$ (Figure 1.26) of all the elements in each grid of the overset mesh system. 2^{nd}step: Build the ADT for each grid,

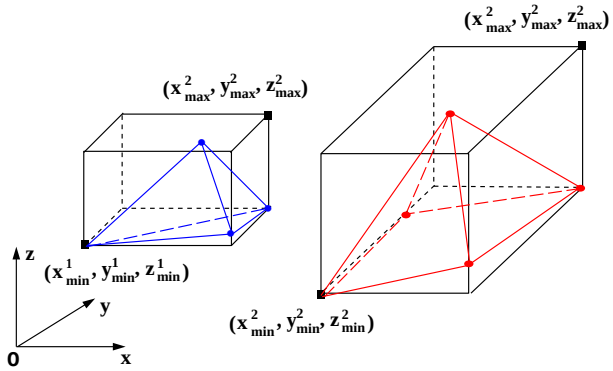


Figure 1.26: Cartesian bounding boxes of tetrahedra and pyramid

the dataset used for building the tree comprises the bounding box coordinates of the grid cells.

3^{rd}step: Search in the tree for a given target point. The algorithm searches the ADT of the given grid and creates a list of cells whose bounding boxes overlap for a given target point I_B see Figure 1.27. To identify the cell that contains the target point, the following criterion

$$min(N_i, 1 - N_i) \geq 0, \quad \forall i \tag{1.1}$$

must be satisfied. N_i are the shape functions well known from finite element methods [3]. They are computed using the formula $N_i(P_j) = \delta_{ij}$, where δ_{ij} is the Kronecker's symbol. Other types

of elements are split into tetrahedrons and the shape functions are computed for each of the subelements. Once a donor cell is identified, linearly interpolate the solution at the nodes of the donor cell e_A to the target point I_B see figure 1.27 as follows :

$$f_{I_B} = \sum_{i_A} f_{i_A} \cdot [N_{i_A}]_{I_B} \quad \text{with} \quad \sum_{i_A} N_{i_A} = 1 \tag{1.2}$$

where f_{i_A} represents the solution at the corner nodes of the element e_A and $[N_{i_A}]_{I_B}$ represents the shape function of the element e_A evaluated at the target point I_B.

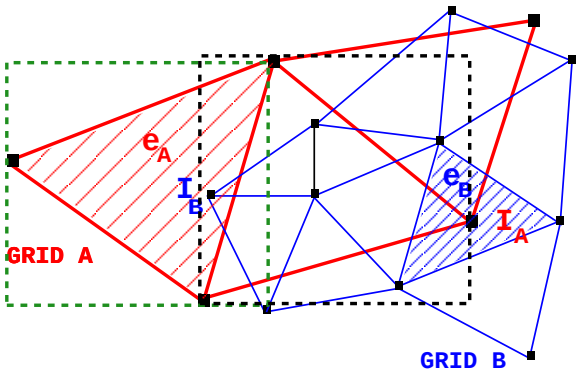


Figure 1.27: Overlapping bounding boxes

1.8.3 Calculations of shape functions.

Linear case :

The interpolation is based on linear shape functions well known from finite element methods. For tetrahedral cells, the linear shape functions are given by the following formulas; $N_1 = 1 - \xi - \eta - \zeta$, $N_2 = \xi$, $N_3 = \eta$ and $N_4 = \zeta$, where (ξ, η, ζ) represent the coordinates in the parametric space see figure 1.28. For other types of elements, hexahedron, prism and

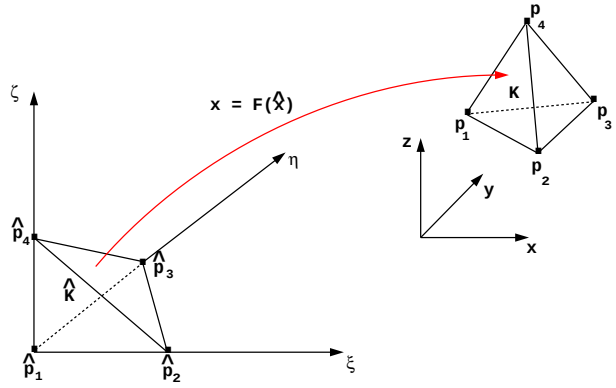


Figure 1.28: Parametric space

pyramid , the interpolation is also based on linear shape functions by splitting those elements into tetrahedrons (see Figures 1.29, 1.30 and 1.31) . Table 1.1 gives all possible subdivisions of the hexahedron p_1 to p_8 into five or six tetrahedra as a function of the number n of diagonals through the vertex p_7.

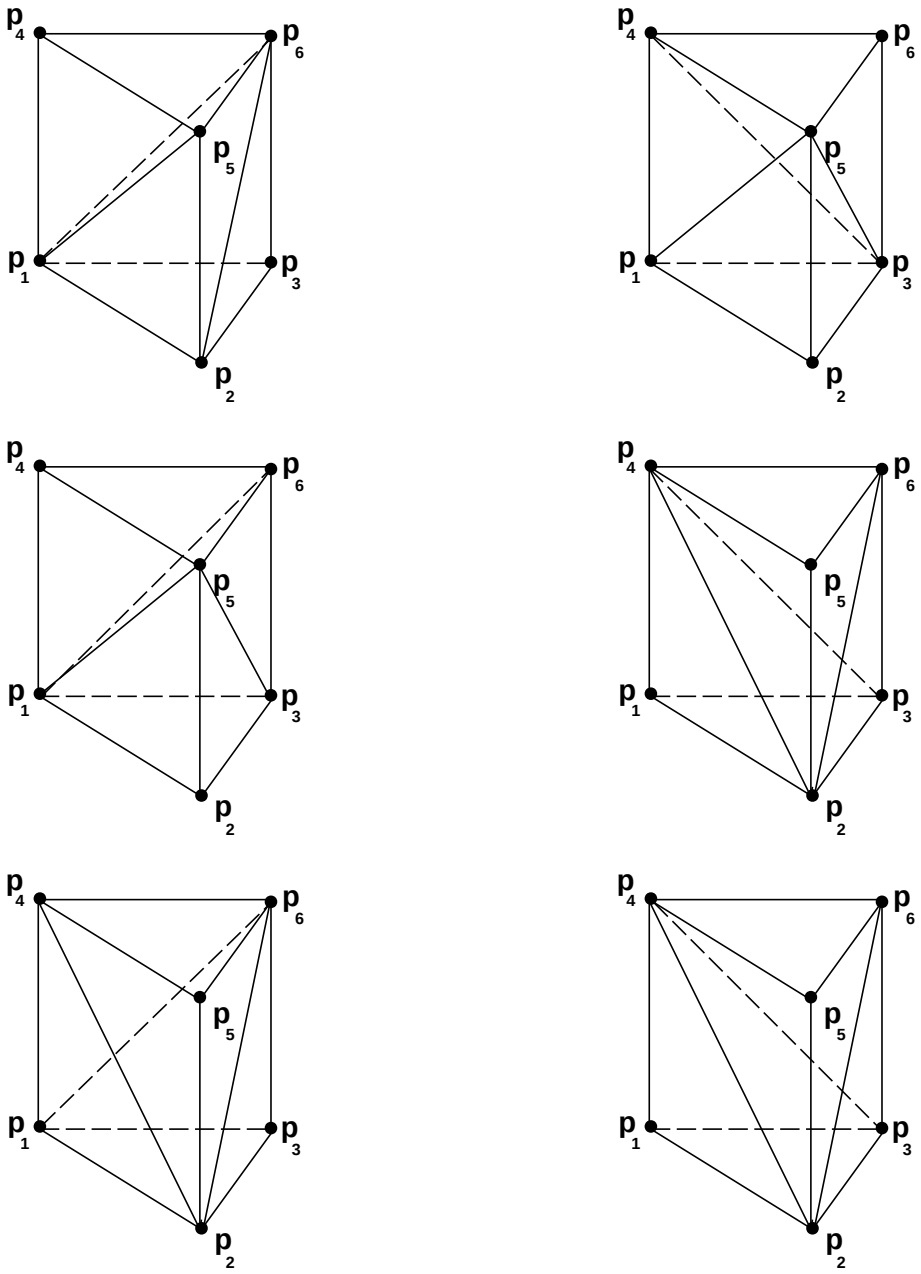


Figure 1.29: The six ways to split the quadrilateral faces of the prism such that it can be subdivided into three tetrahedra

n	$\triangle_1$	$\triangle_2$	$\triangle_3$	$\triangle_4$	$\triangle_5$	$\triangle_6$
0	1,2,3,6	1,3,8,6	1,3,4,8	1,6,8,5	3,8,6,7	Nil
1	1,6,8,5	1,2,8,6	2,7,8,6	1,8,3,4	1,8,2,3	2,8,7,3
	1,6,8,5	1,2,7,6	1,7,8,6	1,8,3,4	1,8,7,3	2,1,7,3
2	1,5,6,7	1,4,8,7	1,8,5,7	1,2,3,6	1,4,7,3	1,7,6,3
	1,3,4,7	1,5,7,8	1,7,4,8	1,2,3,6	1,7,5,6	1,3,7,6
3	1,3,4,7	1,4,8,7	1,8,5,7	1,6,7,5	2,6,7,1	2,7,3,1
	1,2,7,6	1,7,8,5	1,6,7,5	1,7,2,3	1,8,7,4	1,7,3,4
	1,2,7,6	1,2,3,7	1,3,4,7	1,6,7,5	1,7,4,8	1,7,8,5

Table 1.1: All possible subdivision of the hexahedrons into six tetrahedrons.

Nonlinear case :

This subsection describes a method developed for function interpolation within different element types, which is based on the finite element interpolation theory [3].

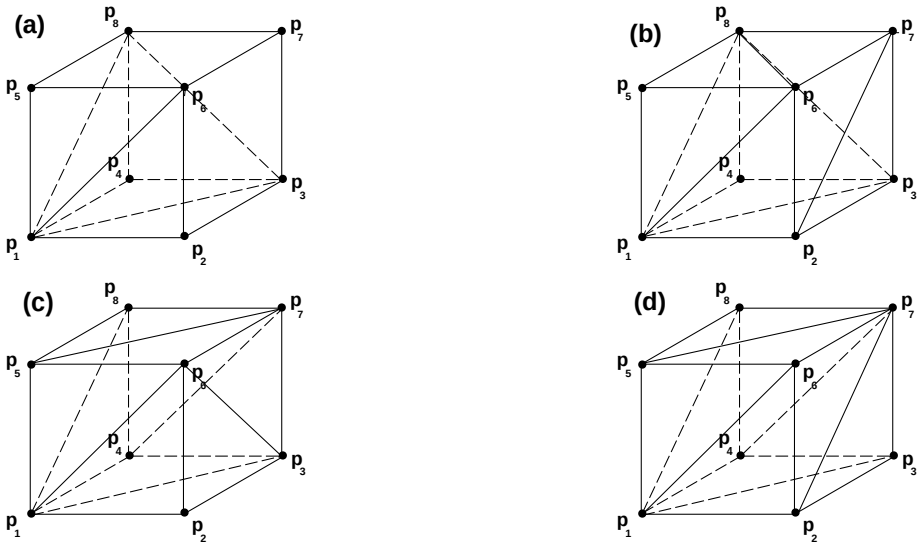


Figure 1.30: First configuration with no diagonal that goes through vertex p_7 (a), second configuration with one diagonal that goes through vertex p_7 (b), third configuration with two diagonals that go through vertex p_7 (c) and fourth configuration with three diagonals that go through vertex p_7 (d)

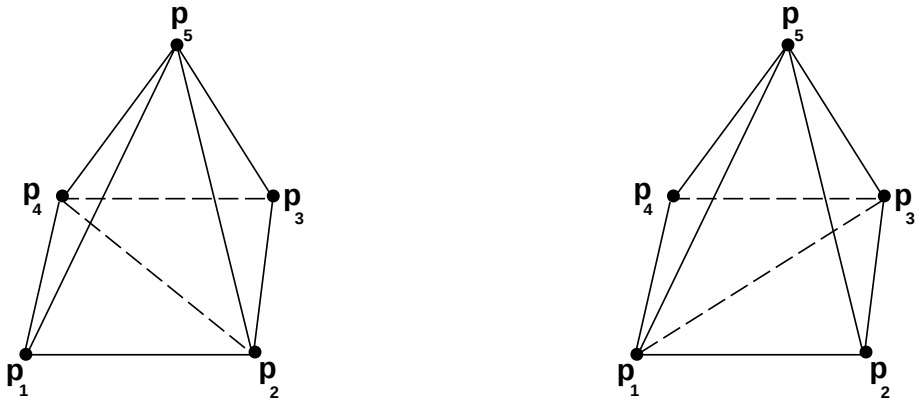


Figure 1.31: The two ways to split the quadrilateral face of the pyramid and to subdivide the pyramid into two tetrahedra

Following this theory, a function F is approximated as :

$$F(\xi,\eta,\zeta) = \sum_i N_i(\xi,\eta,\zeta)F_i$$

where (ξ,η,ζ) represent the coordinates in the parametric space, N_i the shape functions associated to the node i of the element and F_i the value of the function at this node.

Let denote $H = (P_1,P_2,P_3,P_4,P_5,P_6,P_7,P_8)$ denote a hexahedron, and M a point, we obtain $F(M)$ by the following formula:

$$F(M) = \sum_i^8 N_i(\xi_M,\eta_M,\zeta_M)F_i \tag{1.3}$$

where

$$N_1 = 0.125(1-\xi)(1-\eta)(1-\zeta), \quad N_2 = 0.125(1+\xi)(1-\eta)(1-\zeta)$$
$$N_3 = 0.125(1+\xi)(1+\eta)(1-\zeta), \quad N_4 = 0.125(1-\xi)(1+\eta)(1-\zeta)$$

$$N_5 = 0.125(1-\xi)(1-\eta)(1+\zeta), \quad N_6 = 0.125(1+\xi)(1-\eta)(1+\zeta)$$
$$N_7 = 0.125(1+\xi)(1+\eta)(1+\zeta), \quad N_8 = 0.125(1-\xi)(1+\eta)(1+\zeta)$$

To determine the coefficients (ξ_M, η_M, ζ_M), we set $F(M) = \overrightarrow{P_1M}$ to (1.3), and we obtain:

$$\begin{aligned} \overrightarrow{P_1M} &= N_2\overrightarrow{P_1P_2} + N_3\overrightarrow{P_1P_3} + N_4\overrightarrow{P_1P_4} + N_5\overrightarrow{P_1P_5} + \\ &\quad N_6\overrightarrow{P_1P_6} + N_7\overrightarrow{P_1P_7} + N_8\overrightarrow{P_1P_8} \end{aligned} \tag{1.4}$$

Using (1.4), the following function can be built:

$$\begin{aligned} G(\xi_M, \eta_M, \zeta_M) &= N_2\overrightarrow{P_1P_2} + N_3\overrightarrow{P_1P_3} + N_4\overrightarrow{P_1P_4} + \\ &\quad N_5\overrightarrow{P_1P_5} + N_6\overrightarrow{P_1P_6} + N_7\overrightarrow{P_1P_7} + \\ &\quad N_8\overrightarrow{P_1P_8} - \overrightarrow{P_1M} \end{aligned} \tag{1.5}$$

The coefficients (ξ_M, η_M, ζ_M) are solution of $G(\xi_M, \eta_M, \zeta_M) = \overrightarrow{0}$, which is solved using Newton method, i.e.

$$\overrightarrow{\xi_M^{n+1}} = \overrightarrow{\xi_M^n} - G_\xi^{-1} G \tag{1.6}$$

where G_ξ is the Jacobian matrix of the function G. Then, the interpolation weights are computed as $N_i(\overrightarrow{\xi_M})$.

A similar procedure is applied to prisms and pyramids, with the corresponding shape functions

$$N_1 = 0.5(1-\zeta)(1-\xi-\eta), \quad N_2 = 0.5(1-\zeta)\xi, \quad N_3 = 0.5(1-\zeta)\eta$$

$$N_4 = 0.5(1+\zeta)(1-\xi-\eta), \quad N_5 = 0.5(1+\zeta)\xi, \quad N_6 = 0.5(1+\zeta)\eta$$

for prisms and

$$N_1 = 0.25[(1-\xi)(1-\eta)-\zeta], \quad N_2 = 0.25[(1+\xi)(1-\eta)-\zeta]$$

$$N_3 = 0.25[(1+\xi)(1+\eta)-\zeta], \quad N_4 = 0.25[(1-\xi)(1+\eta)-\zeta], \quad N_5 = \zeta$$

for pyramids.

1.8.4 Parallel implementation of overset unstructured grids method

For the parallel implementation of chimera method, a domain decomposion is adopted. At the beginning we have several grid blocks where each grid block is a physical grid with own volume triangulation and boundary description. These grid blocks are not connected in any way. The only communication between these blocks is done by a chimera interpolation. The grid blocks are written into one data structure and afterwards these data structure is partitioned see Figure 1.32. Each partition may have a different number of blocks. There exists a local numbering of the grid blocks within one partition and a mapping from the local block number to the global block number of the non partitioned grid. One of the best domain decomposition for overset grid selected from several cases of division is illustrated in Figure 1.34. The parallel algorithm is explained for a rotating profile is embedded in a background mesh which is shown in Figure 1.34. The grid consists of two blocks and four partitions. The algorithm is explained being on partition 0. This partition is denotiated as own partition. Each partition contains a list of chimera boundaries. Each chimera boundary has a search partition list and a search block list. The lists are shown in Figure 1.33. If no knowledge and no further algorithm is provided the list of search partitions contains all partitions and the list of search blocks contains

all blocks excluding the block of the chimera boundary. To reduce the search effort a global alternating digital tree of the grid partitions can be introduced. With this tree intersections can be detected between a bounding box of a chimera boundary and the bounding boxes of the grid partitions see Figure 1.35. The list of search block should be specified by user input. It has to be determined only once at the beginning whereas the list of search partitiones should be updated each physical time step.For the explanation of the algorithm and the implementation of parallel chimera that has been developed for the DLR TAU-Code see [43, 44].

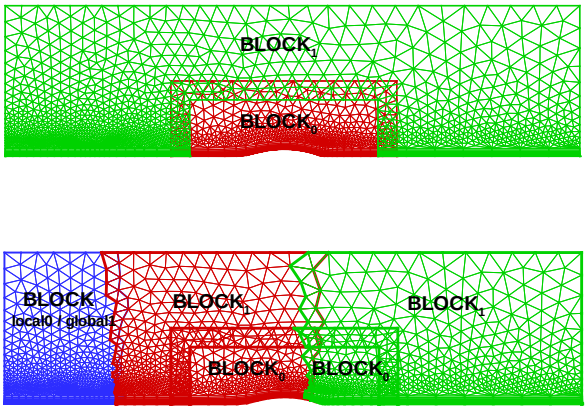


Figure 1.32: Partitioned overset unstructured grids

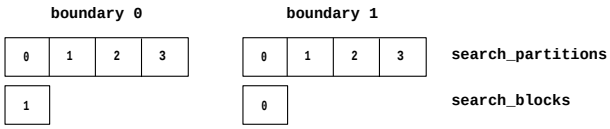


Figure 1.33: Input lists for Chimera search

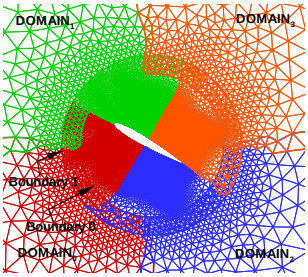


Figure 1.34: Overset grid system and partitioned grid

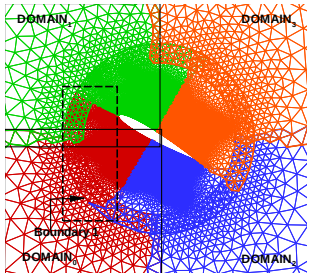


Figure 1.35: A global ADT

2 Flow Solver

2.1 Introduction

The flow solver is a three-dimensional, parallel, hybrid, multigrid code. It implemented a finite volume scheme for solving the Reynolds-averaged Navier-Stokes (RANS) equations. The flow variables are stored on the vertices of the initial grid. This type of spatial discretization is called cell vertex with a dual metric which is computed during the preprocessing step. The code uses explicit time stepping, the multistep Runge-Kutta scheme and optionally implicit time stepping with a LU-time scheme. For accelerating the convergence to steady state a local time-stepping concept, different residual smoothing algorithm and a geometrical multigrid method is implemented.

The computation of the fluxes can be performed with either an upwind or a central scheme. The flux discretization in an upwind scheme can be chosen from several different functions:

- Van_Leer,
- AUSMDV,
- AUSMP,
- Roe,
- AUSM_Van_Leer,
- EFM,
- MAPS+,

The central method is available with two different dissipation models:

- scalar dissipation,
- matrix dissipation,

The viscous fluxes for the one equation turbulence models with central schemes are discretized using central differences. For two equation models the central scheme uses an upwind type discretization for their viscous fluxes.

The coarse dual grids provided by the preprocessor have the same properties as the fine dual grid. Hence the governing equations can be discretized on the coarse grids employing the identical algorithms as on the fine grid. The only difference is that the discretization is switched to first order on the coarse grids when employing an upwind scheme. Furthermore the source

terms of the turbulence model are only calculated on the fine grid and injected onto the coarse grids.

The turbulence model implemented in the solver is the one-equation transport model according to Spalart and Allmaras (SA). The model uses only local quantities for calculating turbulent transport which makes it suitable for unstructured methods.

In order to provide control over laminar regions a modification of the source terms is employed, which ensures that $\nu_t = 0$ (ν_t is the eddy viscosity) is a stable solution. For transition to turbulent flow the modification of the source-term is switched off on points near no-slip walls where turbulent flow is prescribed, whereas it is switched on for points near "laminar walls".

The data structure provided by the preprocessor allows to running the solver in several subdomains in parallel. The only work to be performed in multiprocessor mode additional is the communication required to update the point-data for the 'additional points'. This communication between the different concurrent processes is done using the communication tables and the MPI-library. Since this library supports non-blocking sends and receives, all data to be send is copied into send buffers and all receive/sends are started together. The communication is hidden by an interface routine, such that inside the solver only calls of this interface appear.

For time-accurate solutions a global as well as a dual time-stepping scheme are implemented. The dual time stepping scheme follows the approach of Jameson [32], where the Runge-Kutta scheme is slightly changed in order to avoid instabilities while dealing with small physical time-steps. The time using dual-time discretization can be chosen to be first, second or third order (where a higher order implies increased overhead).

2.2 Governing Equations

The Navier-Stokes equations for the three dimensional case can be written in conservative form as

$$\frac{\partial}{\partial t}\iiint_V \vec{W}\,dV = -\iint_{\partial V} \bar{\bar{F}}\cdot\vec{n}\,dS \tag{2.1}$$

where

$$\vec{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}$$

is the vector of the conserved quantities. V denotes an arbitrary control volume with the boundary ∂V and the outer normal $\vec{n}$. The flux density tensor $\bar{\bar{F}}$ is composed of the flux vectors in the three coordinate directions:

$$\bar{\bar{F}} = (\vec{F}_i^c + \vec{F}_v^c)\cdot\vec{e}_x + (\vec{G}_i^c + \vec{G}_v^c)\cdot\vec{e}_y + (\vec{H}_i^c + \vec{H}_v^c)\cdot\vec{e}_z \tag{2.2}$$

with e_x, e_y and e_z being unit vectors in the coordinate directions. The indices i and v denote the inviscid and the viscous contributions, respectively. The viscous contributions are neglected

when considering the Euler equations. The viscous and the inviscid fluxes are

$$\vec{F}_i^c = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho Hu \end{pmatrix}, \quad \vec{F}_v^c = - \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + \kappa_l \frac{\partial T}{\partial x} \end{pmatrix}, \tag{2.3}$$

$$\vec{G}_i^c = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho Hv \end{pmatrix}, \quad \vec{G}_v^c = - \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{xy} + v\tau_{yy} + w\tau_{yz} + \kappa_l \frac{\partial T}{\partial y} \end{pmatrix}, \tag{2.4}$$

$$\vec{H}_i^c = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho Hw \end{pmatrix}, \quad \vec{H}_v^c = - \begin{pmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{xx} \\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + \kappa_l \frac{\partial T}{\partial z} \end{pmatrix}. \tag{2.5}$$

The pressure is calculated by the equation of state

$$p = (\gamma - 1)\rho(E - \frac{u^2 + v^2 + w^2}{2}). \tag{2.6}$$

From equation 2.1 the temporal change of the conservative variables $\vec{W}$ can be derived as:

$$\frac{\partial}{\partial t}\vec{W} = -\frac{\iint_{\partial V} \bar{\bar{F}} \cdot \vec{n} dS}{\iiint_V dV} \quad . \tag{2.7}$$

The change of the flow conditions in a control volume V is given by the flux over the control volume boundary ∂V related to the size of V. For a control volume fixed in time and space, equation 2.7 can be written as:

$$\frac{d}{dt}\vec{W} = -\frac{1}{V} \cdot \vec{Q}^F \tag{2.8}$$

with $\vec{Q}^F$ representing the fluxes over the boundaries of the control volume. If the boundary is divided into n faces, $\vec{Q}^F$ is given by:

$$\vec{Q}^F = \sum_{i=1}^{n} \vec{Q}_i^F = \sum_{i=1}^{n} (\vec{Q}_i^{F,c} - \vec{D}_i)$$

where $\vec{Q}_i^{F,c}$ denote the inviscid fluxes over the respective face. Hence, in order to determine the temporal change of the flow quantities in a control volume, the convective fluxes over the control volume boundaries have to be determined. For upwind schemes the dissipative terms $\vec{D}_j$ are zero, but for central schemes additional dissipative terms have to be computed.

2.3 Upwind Schemes - Spatial Discretization

The approximation of the governing equations can be done with finite-difference quotients. The partial differential equations involves a number of different partial derivative terms which can be replaced by finite differences. The resulting formula is called a difference equation. This difference equation is an algebraic representation of the partial differential equation. To explain this in more detail we consider the continuity equation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0.$$

If we apply the velocity u as constant, equation 2.3 takes the typical form of a linear convection equation:

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} = 0 \quad \text{or the shorthand} \quad \rho_t + u\rho_x = 0,$$

describing the transport of mass ρ by a flow of velocity u.
We study the flux balance for the dual cell around point i. Let a superscript n denote the time step. Applying a forward difference scheme in time, the corresponding finite volume scheme reads

$$\frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} = -\frac{u}{\Delta x}\left(\rho^n_{i+\frac{1}{2}} - \rho^n_{i-\frac{1}{2}}\right) \tag{2.9}$$

As $u > 0$ and constant we may use a simple upwind scheme, i.e. the flux at $i+\frac{1}{2}$ and $i-\frac{1}{2}$ is approximated by ρ_i and ρ_{i-1} resp. The discrete formulation then reads:

$$\frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} = -\frac{u}{\Delta x}(\underbrace{\rho_i^n}_{\rho^n_{i+\frac{1}{2}}} - \underbrace{\rho_{i-1}^n}_{\rho^n_{i-\frac{1}{2}}}) \tag{2.10}$$

This is a *first-order upwind* scheme.
Now we show that this simple upwind scheme can be written in artificial viscosity form, i.e., as a central scheme with artificial dissipation. If the use a central scheme for the flux at $i+\frac{1}{2}$ and $i-\frac{1}{2}$, then (2.9) becomes

$$\frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} = -\frac{u}{\Delta x}[\underbrace{\frac{1}{2}(\rho_i^n + \rho_{i+1}^n)}_{\rho^n_{i+\frac{1}{2}}} - \underbrace{\frac{1}{2}(\rho_{i-1}^n + \rho_i^n)}_{\rho^n_{i-\frac{1}{2}}}] \tag{2.11}$$

which is of second order accuracy. Subtracting the upwind scheme (2.10) from the central scheme (2.11), we obtain

$$(\rho_i^n - \rho_{i-1}^n) - \left[\frac{1}{2}\left(\rho_i^n + \rho_{i+1}^n\right) - \frac{1}{2}\left(\rho_{i-1}^n + \rho_i^n\right)\right] = -\frac{1}{2}(\rho_{i+1}^n - \rho_i^n) + \frac{1}{2}(\rho_i^n - \rho_{i-1}^n)$$

Thus the upwind scheme can be written as

$$\frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} = -\frac{u}{\Delta x}\left[\frac{1}{2}\left(\rho_i^n + \rho_{i+1}^n\right) - \frac{1}{2}\left(\rho_{i-1}^n + \rho_i^n\right) - \frac{1}{2}(\rho_{i+1}^n - \rho_i^n) + \frac{1}{2}(\rho_i^n - \rho_{i-1}^n)\right].$$

Hence the upwind scheme can be written in *artificial viscosity form* as

$$\frac{\rho_i^{n+1}-\rho_i^n}{\Delta t} = -\frac{u}{\Delta x}[\underbrace{\{\frac{1}{2}\left(\rho_i^n+\rho_{i+1}^n\right)-\frac{1}{2}(\rho_{i+1}^n-\rho_i^n)\}}_{\rho_{i+\frac{1}{2}}^n} - \underbrace{\{\frac{1}{2}\left(\rho_{i-1}^n+\rho_i^n\right)-\frac{1}{2}(\rho_i^n-\rho_{i-1}^n)\}}_{\rho_{i-\frac{1}{2}}^n}]$$

In order to calculate the inviscid fluxes over a control volume face, the Riemann problem has to be solved for the face. In this section the determination of the convective terms employing different types of Riemann solvers is described. The flux computed over a face (Figure 2.1) can be expressed as:

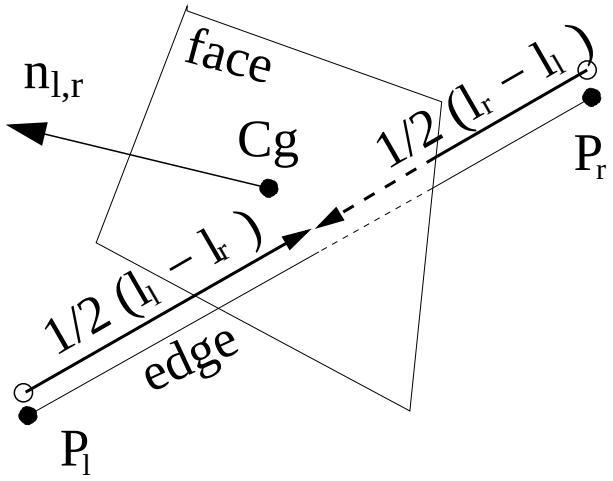


Figure 2.1: Face for an upwind flux computation

$$\begin{aligned} \vec{F}^{face} &= \frac{1}{2}\left(\vec{F}_l+\vec{F}_r\right)-\frac{1}{2}\mid\bar{\bar{A}}\left(\vec{w}_l,\vec{w}_r,\vec{n}_{l,r}\right)\mid(\vec{w}_r-\vec{w}_l), \\ \text{with}\quad \vec{F}_l &= \vec{F}_l\cdot n_x+\vec{G}_l\cdot n_y+\vec{H}_l\cdot n_z, \\ \text{and}\quad \vec{F}_r &= \vec{F}_r\cdot n_x+\vec{G}_r\cdot n_y+\vec{H}_r\cdot n_z, \end{aligned} \tag{2.12}$$

The vector $\vec{\bar{F}}^{face}$ represents the flux density vector normal to the cell interface (face in Figure 2.1), $\bar{A}$ is the corresponding flux Jacobian. The lower indices l,r denote variables to the right and left of the cell face.

For the first order upwind scheme in 2.10 and 2.13 the solution is constant over the control volume. To achieve second order accuracy. the solution is assumed to be piecewise linear over the control volume. Therefor we use a higher order reconstruction for the primitive variables ρ,u,v,w,p. The extrapolation to all faces of the control volume is written as a Taylor expansion (explanatory for ρ):

$$\begin{aligned} \rho_{face,r} &= \rho_{node,r}+\frac{1}{2}\psi\vec{\nabla}\rho_r\left(\vec{l}_l-\vec{l}_r\right), \quad \text{and} \\ \rho_{face,l} &= \rho_{node,l}+\frac{1}{2}\psi\vec{\nabla}\rho_l\left(\vec{l}_r-\vec{l}_l\right), \end{aligned} \tag{2.13}$$

$\vec{l}$ is the location vector of the corresponding points, ρ_{node} is the value one time step ago and the limiter ϕ (see also Part 2.3.5, as proposed by Venkatakrishnan [72], reduces the scheme to first order at discontinuities to avoid oscillations of the gradient. The only unknowns in formula 2.13 are the gradients $\vec{\nabla}\rho$. The computation of the gradients is explained in Part 2.3.3.

2.3.1 AUSM-Scheme

The advection upstream splitting method AUSM was devised by [41]. The DLR TAU-Code implements a modified AUSM scheme proposed by [36]. The aim of [36] is to construct a hybrid scheme which has the robustness of the van Leer scheme in the vicinity of shocks and the low diffusive properties in smooth regions of the AUSM scheme [41].

Let the points $P(j1)$ and $P(j2)$ be separated by the face $F(i)$: $P(j1) = C_F^P(i,1)$ and $P(j2) = C_F^P(i,2)$. The convective fluxes have to be considered as fluxes between the control volumes surrounding the points $P(j1)$ and $P(j2)$. The size and orientation of the face is described by the face normal vector $\vec{F}(i)$, see Figure 2.10. The normalized components of the face vector are derived by:

$$\vec{F}^n(i) = \begin{pmatrix} n^x(i) \\ n^y(i) \\ n^z(i) \end{pmatrix} = \frac{1}{|\vec{F}(i)|} \cdot \vec{F}(i).$$

For first order accurate calculations the flow conditions on the left $\vec{W}_L$ and the right $\vec{W}_R$ hand side of face $F(i)$ can be taken directly from the respective points, since the flow conditions in this case are assumed to be constant over the control volume:

$$\vec{W}_L = \vec{W}(j1),$$

and

$$\vec{W}_R = \vec{W}(j2).$$

For a second order accurate calculation the flow conditions have to be extrapolated to the face employing the gradients of the flow variables and the vector pointing from the points to the face. As the face crosses the edge between the two points $P(j1)$ and $P(j2)$ halfway, the vector

$$\vec{d} = 0.5 \cdot (\vec{P}(j2) - \vec{P}(j1))$$

is pointing from $P(j1)$ to the face $F(i)$ and the vector $(-\vec{d})$ from $P(j2)$ to the face.

The gradients

$$\nabla\vec{u}(j1) = \begin{pmatrix} u_x(j1) \\ u_y(j1) \\ u_z(j1) \end{pmatrix}$$

with $\{\nabla\vec{u}(m) \,|\, m = 1,\ldots N^P\}$ of a flow variable u for $P(j1)$ have to be scaled by a limiting factor $\Theta_u(j1)$ with $\{\Theta_u(m) \,|\, m = 1,\ldots N^P\}$ in order to enforce a monotone behavior of the reconstructed values. The evaluation of this factor is described in Part 2.3.5.

A variable u can now be reconstructed on the left hand side of face $F(i)$ as

$$u_L = u(j1) + \vec{d} \cdot \nabla\vec{u}(i) \cdot \Theta_u(j1). \tag{2.14}$$

The extrapolation results in two vectors of primitive variables

$$\vec{U}_L = \begin{pmatrix} \rho \\ u \\ v \\ w \\ p \end{pmatrix}_L \quad \text{and} \quad \vec{U}_R = \begin{pmatrix} \rho \\ u \\ v \\ w \\ p \end{pmatrix}_R,$$

for the left and for the right hand side of the face. From the variable vectors additional values are computed for the left and the right side. The contravariant Mach number for the left hand side can be computed as:

$$M_L = \frac{vn_L}{a_L}, \tag{2.15}$$

with

$$vn_L = \begin{pmatrix} u \\ v \\ w \end{pmatrix}_L \cdot \begin{pmatrix} n^x(i) \\ n^y(i) \\ n^z(i) \end{pmatrix} \quad \text{and} \quad a_L = \sqrt{\frac{\gamma \cdot p_L}{\rho_L}}.$$

The enthalpy H_L can be determined as:

$$H_L = \frac{p_L \cdot \gamma}{\rho_L \cdot (\gamma - 1)} + \frac{u_L^2 + v_L^2 + w_L^2}{2} \tag{2.16}$$

The respective values for the right hand side are computed accordingly.

From the contravariant Mach numbers M_L and M_R on both sides of the face the advection Mach number M_F at the cell face is calculated as:

$$M_F = M_L^p + M_R^m \tag{2.17}$$

where the split Mach numbers $M^{p/m}$ are defined as

$$M^p = \begin{cases} M & \text{if } M \geq 1 \\ \frac{1}{4}(M+1)^2 & \text{if } |M| < 1 \\ 0 & \text{if } M \leq -1 \end{cases},$$

$$M^m = \begin{cases} 0 & \text{if } M \geq 1 \\ -\frac{1}{4}(M-1)^2 & \text{if } |M| < 1 \\ M & \text{if } M \leq -1 \end{cases}.$$

Herein M denotes the Mach number of the flow normal to the cell face (M_L or M_R).

The pressure p_F at the cell face is calculated in a similar way as

$$p_F = p_L^p + p_R^m \tag{2.18}$$

where $p^{p/m}$ denote the split pressure

$$p^p = \begin{cases} p & \text{if } M \geq 1 \\ \frac{1}{4} \cdot p \cdot (M+1)^2 \cdot (2-M) & \text{if } |M| < 1 \\ 0 & \text{if } M \leq -1 \end{cases},$$

$$p^m = \begin{cases} 0 & \text{if } M \geq 1 \\ \frac{1}{4} \cdot p \cdot (M-1)^2 \cdot (2+M) & \text{if } |M| < 1 \\ p & \text{if } M \leq -1 \end{cases}.$$

The dissipation of the scheme is controlled by the coefficient Φ_F. In the original scheme of Liou, Φ_F is set to

$$\Phi_F = |M_F| \quad .$$

The aim of [36] was to combine the van Leer scheme and the AUSM scheme of Liou and Steffen using the hybrid form

$$\Phi_F = (1-\omega)\Phi_F^{VL} + \omega \Phi_F^{modAUSM} \quad .$$

with

$$\Phi_F^{VL} = \begin{cases} |M_F| & \text{if} \quad |M_F| \geq 1 \\ |M_F| + \frac{1}{2}(M_R - 1)^2 & \text{if} \quad 0 \leq M_F < 1 \\ |M_F| + \frac{1}{2}(M_L + 1)^2 & \text{if} \quad -1 < M_F \leq 0 \end{cases},$$

$$\Phi_F^{modAUSM} = \begin{cases} |M_F| & \text{if} \quad |M_F| > d_{cut} \\ \frac{|M_F|^2 + d_{cut}^2}{2 d_{cut}} & \text{if} \quad |M_F| \leq d_{cut} \end{cases}.$$

where d_{cut} is a small parameter $0 \leq d_{cut} \leq 0.5$, which is introduced to ensure a non-vanishing artificial dissipation if M_F becomes small. In order to switch between the robust van Leer scheme in the vicinity of shocks and the AUSM scheme with its low diffusive properties in smooth regions, the following shock-switch ω is used

$$\omega = \min(\nu_L, \nu_R)$$

The shock-switch ω is based on the second difference of the pressure by

$$\nu_L = \max\left(1 - \alpha \frac{|p_{LL} - 2p_L + p_R|}{|p_{LL} + 2p_L + p_R|} \;,\; 0\right)$$
$$\nu_R = \max\left(1 - \alpha \frac{|p_L - 2p_R + p_{RR}|}{|p_L + 2p_R + p_{RR}|} \;,\; 0\right) \quad .$$

The value of ω is one in smooth regions and switches to zero in the vicinity of shocks.
From the obtained information, the inviscid flux $\vec{Q}_F^{F,c}$ over Face $F(i)$ can now be calculated as:

$$\vec{Q}_F^{F,c} = \frac{1}{2} \cdot |\vec{F}(i)| \cdot \left[M_F \cdot \left[\begin{pmatrix} \rho a \\ \rho a \cdot u \\ \rho a \cdot v \\ \rho a \cdot w \\ \rho a H \end{pmatrix}_L + \begin{pmatrix} \rho a \\ \rho a \cdot u \\ \rho a \cdot v \\ \rho a \cdot w \\ \rho a H \end{pmatrix}_R \right] - \Phi_F \cdot \left[\begin{pmatrix} \rho a \\ \rho a \cdot u \\ \rho a \cdot v \\ \rho a \cdot w \\ \rho a H \end{pmatrix}_L - \begin{pmatrix} \rho a \\ \rho a \cdot u \\ \rho a \cdot v \\ \rho a \cdot w \\ \rho a H \end{pmatrix}_R \right] \right] + \begin{pmatrix} 0 \\ F^x(i) \cdot p_F \\ F^y(i) \cdot p_F \\ F^z(i) \cdot p_F \\ 0 \end{pmatrix} \tag{2.19}$$

After having determined the fluxes between the control volumes surrounding $P(j1)$ and $P(j2)$ they are added to the fluxes $\vec{Q}_{old}^{F,c}(j1)$ and $\vec{Q}_{old}^{F,c}(j2)$ of the two points that have been computed before. As depicted in Figure 2.2 the normal vector $\vec{F}(i)$ is pointing from the face to the direction of point $P(j2)$. Therefore, the calculated fluxes have to be reversed before they can be added to the already obtained fluxes for point $P(j2)$:

$$\vec{Q}^{F,c}(j1) = \vec{Q}_{old}^{F,c}(j1) + \vec{Q}_F^{F,c},$$

and

$$\vec{Q}^{F,c}(j2) = \vec{Q}_{old}^{F,c}(j2) - \vec{Q}_F^{F,c}.$$

2.3.2 AUSMDV-Scheme

The AUSMDV scheme devised by Wada and Liou in [75] provides approximate solutions to the Riemann problem, in the form of a numerical flux. As input, the AUSMDV requires two states

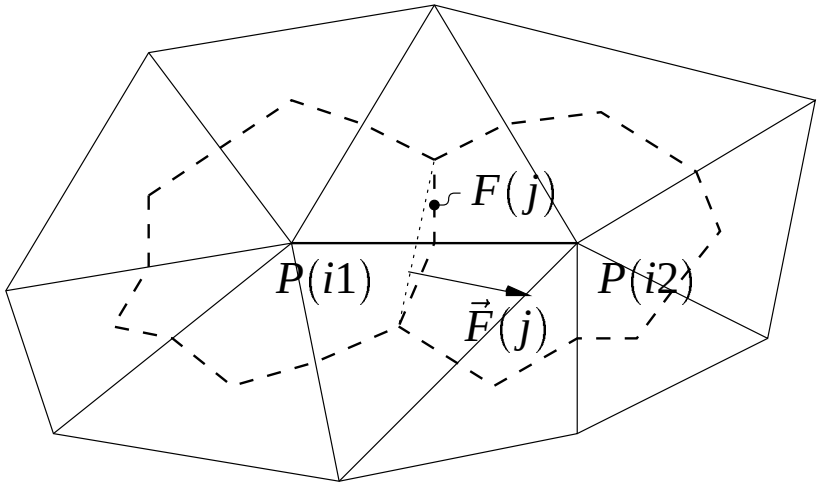


Figure 2.2: Control volumes around neighboring points $P(j1)$ and $P(j2)$

(left and right), the area of the face separating these states, and the unit normal vector $\vec{n}$ of this face pointing from the left state to the right state.

The basic concept is to determine an accurate mass flux ρu through the face, and to convect the properties of state $\vec{\Psi} = (1, vx_t, vy_t, vz_t, H)^T$ from the left or right state, according to the sign of ρu. Here, ρ is the density, $u = \vec{v}\vec{n}$ the normal component of the velocity v, $\vec{v}_t = \vec{v} - u\vec{n}$ the tangential components of the velocity, and H the total enthalpy.

The flux can be written as:

$$\vec{Q} = area \times \begin{cases} (\vec{p}_{\frac{1}{2}} + (\rho u)_{\frac{1}{2}} \vec{\Psi}_l) & ; \quad (\rho u)_{\frac{1}{2}} \geq 0 \\ (\vec{p}_{\frac{1}{2}} + (\rho u)_{\frac{1}{2}} \vec{\Psi}_r) & ; \quad (\rho u)_{\frac{1}{2}} < 0 \end{cases}$$

$$\text{with} \qquad \vec{p}_{\frac{1}{2}} = (0, ((\rho u^2)_{\frac{1}{2}} + p_{\frac{1}{2}})nx, ((\rho u^2)_{\frac{1}{2}} + p_{\frac{1}{2}})ny, ((\rho u^2)_{\frac{1}{2}} + p_{\frac{1}{2}})nz, 0)^T$$

In this flux formula, we have to specify $p_{\frac{1}{2}}$, $(\rho u)_{\frac{1}{2}}$, and $(\rho u^2)_{\frac{1}{2}}$ where subscript $\frac{1}{2}$ denotes an intermediate value at the face. In a manner similar to the van Leer splitting, they are given by:

$$(\rho u)_{\frac{1}{2}} = u_l^+ \rho_l + u_r^- \rho_r \quad \text{and} \quad p_{\frac{1}{2}} = p_l^+ + p_r^- .$$

The pressure splitting p_l^+, p_r^- and velocity splitting u_l^+, u_r^- are defined with a common speed of

sound $c_{l,r} = \max(c_l, c_r)$:

$$p_l^+ = \begin{cases} p_l & ; \quad c_{l,r} \le u_l \\ \frac{p_l}{4}\left(\frac{u_l}{c_{l,r}}+1\right)^2\left(2-\frac{u_l}{c_{l,r}}\right) & ; \quad -c_{l,r} \le u_l \le c_{l,r} \\ 0 & ; \quad u_l \le -c_{l,r} \end{cases},$$

$$p_r^- = \begin{cases} 0 & ; \quad c_{l,r} \le u_r \\ \frac{p_r}{4}\left(\frac{u_r}{c_{l,r}}-1\right)^2\left(2+\frac{u_r}{c_{l,r}}\right) & ; \quad -c_{l,r} \le u_r \le c_{l,r} \\ p_r & ; \quad u_r \le -c_{l,r} \end{cases},$$

$$u_l^+ = \begin{cases} u_l & ; \quad c_{l,r} \le u_l \\ \alpha_l\left[\frac{(u_l+c_{l,r})^2}{4c_{l,r}}-u_l\right]+u_l & ; \quad 0 \le u_l \le c_{l,r} \\ \alpha_l\frac{(u_l+c_{l,r})^2}{4c_{l,r}} & ; \quad -c_{l,r} \le u_l \le 0 \\ 0 & ; \quad u_l \le -c_{l,r} \end{cases},$$

$$u_r^- = \begin{cases} 0 & ; \quad c_{l,r} \le u_r \\ -\alpha_r\frac{(u_r-c_{l,r})^2}{4c_{l,r}} & ; \quad 0 \le u_r \le c_{l,r} \\ -\alpha_r\left[\frac{(u_r-c_{l,r})^2}{4c_{l,r}}+u_r\right]+u_r & ; \quad -c_{l,r} \le u_r \le 0 \\ u_r & ; \quad u_r \le -c_{l,r} \end{cases},$$

with

$$\alpha_l = \frac{2\left(\frac{p}{\rho}\right)_l}{\left(\frac{p}{\rho}\right)_l+\left(\frac{p}{\rho}\right)_r} \quad \text{and} \quad \alpha_r = \frac{2\left(\frac{p}{\rho}\right)_r}{\left(\frac{p}{\rho}\right)_l+\left(\frac{p}{\rho}\right)_r}.$$

The AUSMD and AUSMV variants differ in the handling of the normal momentum convection $(\rho u^2)_{\frac{1}{2}}$. Explicitly:

$$(\rho u^2)_{\frac{1}{2}}^{\text{AUSMV}} = u_l^+(\rho u)_l + u_r^-(\rho u)_r$$

is used for AUSMV, and for AUSMD:

$$(\rho u^2)_{\frac{1}{2}}^{\text{AUSMD}} = (\rho u)_{\frac{1}{2}}\begin{cases} u_l & ; \quad (\rho u)_{\frac{1}{2}} \ge 0 \\ u_r & ; \quad (\rho u)_{\frac{1}{2}} \le 0 \end{cases}.$$

The AUSMDV uses a combination of both variants:

$$(\rho u^2)_{\frac{1}{2}}^{\text{AUSMDV}} = \left(\frac{1}{2}+s\right)(\rho u^2)_{\frac{1}{2}}^{\text{AUSMV}} + \left(\frac{1}{2}-s\right)(\rho u^2)_{\frac{1}{2}}^{\text{AUSMD}}$$

where

$$s = \frac{1}{2}\min\left(1, 10\frac{|p_r-p_l|}{\min(p_l,p_r)}\right).$$

In addition, the AUSMDV includes two treatments for special situations:

- An entropy fix is introduced to prevent a glitch in the solution at sonic points inside simple expansion fans. Therefore, if $(u-c)_l < 0$ and $(u-c)_r > 0$ or $(u+c)_l < 0$ and $(u+c)_r > 0$ the flux is corrected with:

$$\vec{Q}_{\text{efix}} = \vec{Q} - \frac{area}{8}\Delta\lambda(\rho_r\vec{\Psi}_r - \rho_l\vec{\Psi}_l)$$

  where $\Delta\lambda = (u-c)_r - (u-c)_l$ or $\Delta\lambda = (u+c)_r - (u+c)_l$ respectively.

- Like many other Riemann solvers the AUSMDV suffers slightly from the carbuncle phenomenon. To prevent this the location of shocks in the computational domain are detected by converging characteristics, and a more stable Riemann solver is used there. A more diffusive solver, the van Leer splitting modified according to Hänel and Schwane (cf. [37]), is used:

$$\vec{Q}_{\text{Hänel}} = \rho_l u_l^+\vec{\Psi}_l + \rho_r u_r^-\vec{\Psi}_r + \vec{p}_{\frac{1}{2}},$$

  where

$$u^{\pm} = \begin{cases} \pm\frac{1}{4c}(u\pm c)^2 & ; \quad |u| \le c \\ \frac{u\pm|u|}{2} & ; \quad \text{else} \end{cases}$$

$$p^{\pm} = \begin{cases} \frac{p}{4}\left(\frac{u}{c}\pm 1\right)^2(2\mp\frac{u}{c}) & ; \quad |u| \le c \\ p\frac{u\pm|u|}{2u} & ; \quad \text{else} \end{cases}.$$

2.3.3 Higher Order Reconstruction

For second order accurate upwind calculations the gradients of the flow variables have to be determined. The determination results in an array $\{\nabla\vec{u}(j) \mid j = m, \ldots N^P\}$ for each flow quantity u with

$$\nabla\vec{u}(j) = \begin{pmatrix} u_x(j) \\ u_y(j) \\ u_z(j) \end{pmatrix}.$$

There are numerous algorithms to compute the gradients. In the following section three gradient calculations are explained which can be chosen through the parameter file.

1. `Green-Gauss divergence theorem`: *Green_Gauss*
   For a scalar field u the Gauss theorem for the dual cell $B(j1)$ around point $P(j1)$ states that

$$\int_{B(j1)} \frac{\partial u}{\partial x_k} dx = \int_{\partial B(j1)} u\vec{e}_k \cdot \vec{n} d\sigma \tag{2.20}$$

   where e_k denotes the unit vector in the x_k-coordinate direction. In vector notation this reads

$$\int_{B(j1)} \vec{\nabla} u dx = \int_{\partial B(j1)} u\vec{n} d\sigma \tag{2.21}$$

   Since $\vec{\nabla}u$ is assumed to be constant on each dual grid cell, we obtain

$$\vec{\nabla}u = \frac{1}{V(j1)}\int_{B(j1)} \vec{\nabla} u dx = \frac{1}{V(j1)}\int_{\partial B(j1)} u\vec{n} d\sigma \tag{2.22}$$

where $V(j1) = Vol(B(j1))$ is the volume of the dual cell around $P(j1)$.
Then, numerically, the gradient vector $\nabla\vec{u}(j1)$ of u in point $P(j1)$ with $n-1$ neighboring points $P(j2\ldots jn)$ is obtained by employing the following discrete Green-Gauss formula:

$$\nabla\vec{u}(j1) = \frac{1}{V(j1)} \cdot \sum_{k=2}^{n} \frac{1}{2} \cdot (u(j1) + u(jk)) \cdot \vec{F}(i), \tag{2.23}$$

where $V(j1)$ is the volume of the dual cell around $P(j1)$ and $\vec{F}(i)$ is the normal vector of the dual mesh face $F(i)$ dividing the control volumes around $P(j1)$ and $P(jk)$ as shown in figure 2.3.

The evaluation of gradients can be expressed as a summation of contributions from the

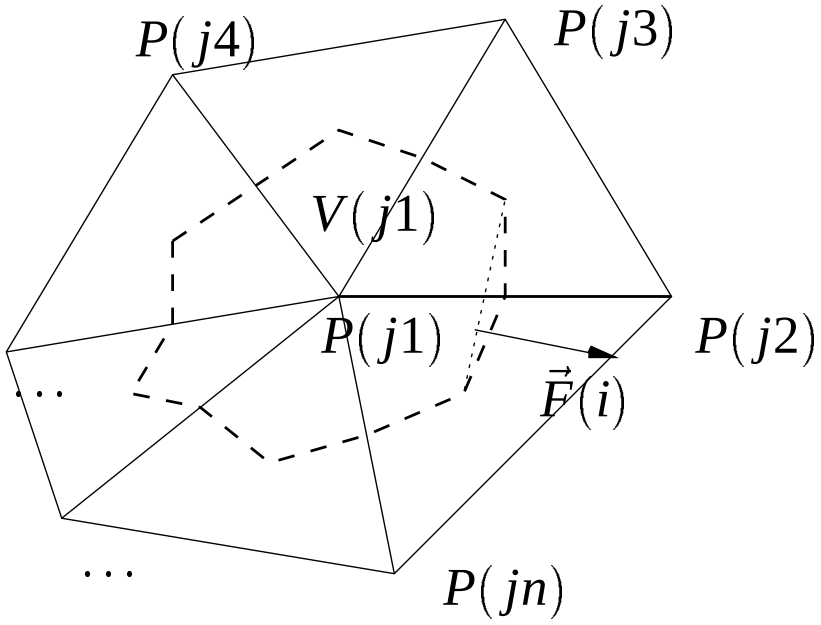


Figure 2.3: Control volume of point $P(j1)$ with neighboring points

grid faces to the related grid points. Consider a face $F(i)$ with the two points $C_F^P(i,1) = P(j1)$ and $C_F^P(i,1) = P(j2)$. The contributions of face $F(i)$ to the gradient vectors of both points can be calculated as the product of the average value of u with the normal vector:

$$\Delta_F\left(\nabla\vec{u}(j1) \cdot V(j1)\right) = \Delta_F\left(\nabla\vec{u}(j2) \cdot V(j2)\right) = \vec{F}(i) \cdot \frac{u(j1) + u(j2)}{2}. \tag{2.24}$$

The resulting vector is added to the actual gradient vectors of the variable u for the points $P(j1)$ and $P(j2)$. As the normal vector is pointing from the face in the direction of $P(j2)$, the contribution for point $P(j2)$ has to be multiplied with -1:

$$\begin{aligned} \nabla\vec{u}(j1) \cdot V(j1) &= (\nabla\vec{u}(j1) \cdot V(j1))_{old} + \Delta_F(\nabla\vec{u}(j1) \cdot V(j1)) \\ \nabla\vec{u}(j2) \cdot V(j2) &= (\nabla\vec{u}(j1) \cdot V(j2))_{old} - \Delta_F(\nabla\vec{u}(j2) \cdot V(j2)) \end{aligned}$$

Additional contributions are to be calculated for points located on the boundaries of the domain. As the value of u is assumed to be constant over the entire boundary face, the contributions for the boundary face $F_b(i)$ of the boundary part b in the three coordinate directions are determined by multiplying the value of the variable $u(j1)$, where $C_{F_b}^P(b,i) = P(j1)$ with the normal vector $\vec{F}_b(i)$:

$$\Delta_{F_b}(\nabla\vec{u}(j1) \cdot V(j1)) = \vec{F}_b(i) \cdot u(i). \tag{2.25}$$

The resulting contributions $\nabla\vec{u}(j1)$ for the point $P(j1)$ are added to the actual gradient vector $\nabla\vec{u}(j1)$. As the normal vector for boundary faces is always pointing to the outside of the domain, the sign of the resulting vector does not have to be changed:

$$\nabla\vec{u}(j1)\cdot V(j1) = (\nabla\vec{u}(j1)\cdot V(j1))_{old} + \Delta_{F_b}(\nabla\vec{u}(j1)\cdot V(j1))$$

After the gradients are determined in that way the obtained vectors are divided by the size $V(j1)$ of the control volume surrounding the point $P(j1)$:

$$\nabla\vec{u}(j1) = \frac{\nabla\vec{u}(j1)\cdot V(j1)}{V(j1)}. \tag{2.26}$$

This approach is only accurate for reconstructing linear functions at triangular meshes (2D) or tetrahedron meshes (3D). Any mixed grid produces considerable errors at interfaces from differing element types.

2. `Least Square with QR decomposition and Gram-Schmidt orthogonalization:` *Least_square*
It was first described by Anderson, Bonhaus [2] and Haselbacher, Blazek [31]. This algorithm uses Taylor expansion from a local point to each surrounding point instead of metric terms as face normals or volumes of a local control volume as it is used for Green-Gauss. A function ϕ expressed by a Taylor expansion for a local point including its neighboring point is:

$$\phi_i = \phi_0 + \frac{\partial\phi_0}{\partial x}\Delta x + \frac{\partial\phi_0}{\partial y}\Delta y + \frac{\partial\phi_0}{\partial z}\Delta z + O(\Delta x^2)\cdots \qquad \phi\ldots\text{arbitrary function} \tag{2.27}$$

The system of linear equations derived from all neighboring points Figure 2.4 can be

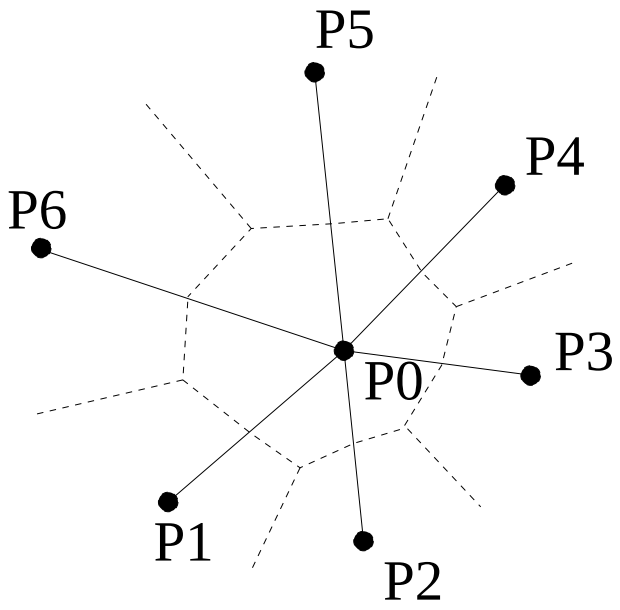


Figure 2.4: Surrounding points used for the least square algorithm

expressed as follows:

$$\underbrace{\begin{bmatrix} w_1\Delta x_{01} & w_1\Delta y_{01} & w_1\Delta z_{01} \\ w_2\Delta x_{02} & w_2\Delta y_{02} & w_2\Delta z_{02} \\ \vdots & \vdots & \vdots \\ w_N\Delta x_{0N} & w_N\Delta y_{0N} & w_N\Delta z_{0N} \end{bmatrix}}_{WA} \underbrace{\begin{pmatrix} \frac{\partial\phi}{\partial x} \\ \frac{\partial\phi}{\partial y} \\ \frac{\partial\phi}{\partial z} \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} w_1(\phi_1-\phi_0) \\ w_2(\phi_2-\phi_0) \\ \vdots \\ w_N(\phi_N-\phi_0) \end{pmatrix}}_{Wb}, \tag{2.28}$$

bzw.

$$W\left[\ \vec{a}_1\,\vec{a}_2\,\vec{a}_3\ \right]\nabla\phi = W\,b \quad \text{with } W = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_N \end{pmatrix}. \tag{2.29}$$

The introduction of a weighting factor which is related to the geometry allows the computation at very high cell aspect ratios. The weighting factor is defined as

$$w_i = \frac{1}{\sqrt{\Delta x_i^2 + \Delta y_i^2 + \Delta z_i^2}} \qquad i = 0,\dots,N \tag{2.30}$$

The solution of matrix A involves a QR decomposition with a Gram-Schmidt orthogonalization. Q is an orthogonal matrix $Q \in \mathbf{R}^{n\times m}$ and R is an upper triangular matrix $R \in \mathbf{R}^{m\times m}$. This may be written as

$$\begin{aligned} Ax &= b \qquad \text{with A} \rightarrow \quad A = QR \\ QRx &= b \qquad \text{multiplied by the transposed} \quad Q^T \\ Q^TQRx &= Q^Tb \qquad \text{with} \quad Q^TQ = I, \quad I \in \mathbf{R}^{m\times m} \\ x &= R^{-1}Q^Tb \end{aligned}$$

and the matrix entities are defined as:

$$A = [\vec{a}_1\,\vec{a}_2\,\vec{a}_3], \quad Q = [\vec{q}_1\,\vec{q}_2\,\vec{q}_3], \quad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix}$$

The entries of the matrix R can now be computed:

$$\begin{aligned} r_{11} &= \sqrt{\sum_i^N (\Delta x_{0i})^2}, \qquad r_{12} = \frac{1}{r_{11}}\sum_i^N (\Delta x\Delta y) \\ r_{13} &= \frac{1}{r_{11}}\sum_i^N (\Delta x\Delta z), \qquad r_{22} = \sqrt{\sum_i^N (\Delta y)^2 - r_{12}^2} \\ r_{23} &= \frac{1}{r_{22}}\sum_i^N (\Delta y\Delta z) - \frac{r_{12}}{r_{11}r_{22}}\sum_i^N (\Delta x\Delta z) \\ r_{33} &= \sqrt{\sum_i^N (\Delta z)^2 - (r_{23}^2 + r_{13}^2)} \end{aligned} \tag{2.31}$$

The computation of gradients with the least-square approach has shown much more accurate gradients as in comparison to the Green-Gauss theorem. Using the achieved better robustness and more accurate solutions as the Green-Gauss approach. This viable algorithm reconstructs linear functions exactly on any type of mixed grids.

3. `Line reconstruction for structured parts:` *Use line reconstruction (0/1)*
   Per default (value 1) the left and right states of the Riemann problem between faces of high aspect ratio cells are reconstructed along lines which will be extracted from the grid. For testing purposes this can be switched off to use the computation of gradients uniformly, based on Green-Gauss or Least Square method. The gradients are stored at the grid points. But this parameter *Use line reconstruction (0/1)* is not available in a release version of TAU.

line2side: Whenever at a local face a neighboring face on each side can be found it is called line2side, Figure 2.5. This allows a second order reconstruction of the gradients.

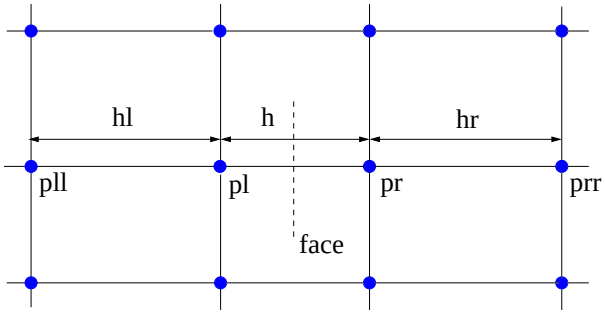


Figure 2.5: Face with neighboring faces on each side

Applying a Taylor expansion for point p_r towards each direction, two expansions are derived which can be written as:

$$\begin{aligned} \phi_l &= \phi_r - \left.\frac{\partial\phi}{\partial x}\right|_r h + \frac{1}{2}\left.\frac{\partial^2\phi}{\partial x^2}\right|_r h^2 + H.O.T. \\ \phi_{rr} &= \phi_r + \left.\frac{\partial\phi}{\partial x}\right|_r h_r + \frac{1}{2}\left.\frac{\partial^2\phi}{\partial x^2}\right|_r {h_r}^2 + H.O.T.. \end{aligned}$$

The resulting formula for the gradient at point p_r becomes:

$$\left.\frac{\partial\phi}{\partial x}\right|_{p_r} = \frac{(\phi_{rr}-\phi_r)\,h^2 + (\phi_r-\phi_l)\,h_r^2}{h\,h_r\,(h+h_r)} \tag{2.32}$$

Applying a Taylor expansion for point p_l in the same manner as before again two formulas are derived as:

$$\begin{aligned} \phi_r &= \phi_l + \left.\frac{\partial\phi}{\partial x}\right|_l h + \frac{1}{2}\left.\frac{\partial^2\phi}{\partial x^2}\right|_l h^2 + H.O.T. \\ \phi_{ll} &= \phi_l - \left.\frac{\partial\phi}{\partial x}\right|_l h_l + \frac{1}{2}\left.\frac{\partial^2\phi}{\partial x^2}\right|_l {h_l}^2 + H.O.T.. \end{aligned}$$

The resulting formula for the gradient at point p_l becomes:

$$\left.\frac{\partial\phi}{\partial x}\right|_{p_l} = \frac{(\phi_l-\phi_{ll})\,h^2 + (\phi_r-\phi_l)\,h_l^2}{h\,h_l\,(h+h_l)} \tag{2.33}$$

line1side: At boundaries the search criterion fails and no *line2side* faces can be found. For example no face can be found left of point p_l. This face is called *line1side*. To have still second order reconstruction the next neighboring point next to p_r - point p_{rr}

can be used for a Taylor expansion Figure 2.6. The second order reconstruction can now be derived as:

$$\begin{aligned}\phi_r &= \phi_l + \frac{\partial\phi}{\partial x}h + \frac{1}{2}\frac{\partial^2\phi}{\partial x^2}h^2 + H.O.T.\\ \phi_{rr} &= \phi_l + \frac{\partial\phi}{\partial x}(h+h_r) + \frac{1}{2}\frac{\partial^2\phi}{\partial x^2}(h+h_r)^2 + H.O.T..\end{aligned}$$

The resulting formula for the gradient becomes:

$$\left.\frac{\partial\phi}{\partial x}\right|_{p_l} = \frac{(\phi_l-\phi_{rr})h^2 + (\phi_r-\phi_l)(h+h_r)^2}{h(h+h_r)h_r}. \tag{2.34}$$

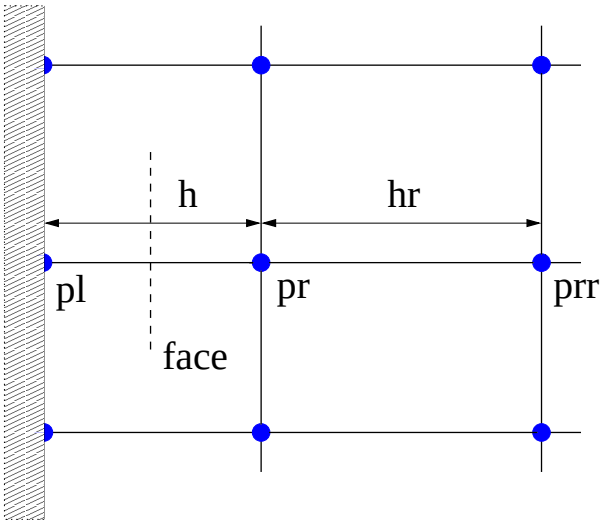


Figure 2.6: Face with one neighboring face on the right side

line0side: On the other hand, there are boundaries where no points exists at the right side of a local face, see Figure 2.7. The second order computation can again be derived from a Taylor expansion using the next neighboring point of p_l - point p_{ll}. The second order reconstruction can now be arrived including the abbreviation $-h = x_l - x_r$ and $-h_l = x_{ll} - x_l$:

$$\begin{aligned}\phi_l &= \phi_r - \frac{\partial\phi}{\partial x}h + \frac{1}{2}\frac{\partial^2\phi}{\partial x^2}h^2 + H.O.T.\\ \phi_{ll} &= \phi_r - \frac{\partial\phi}{\partial x}(h+h_l) + \frac{1}{2}\frac{\partial^2\phi}{\partial x^2}(h+h_l)^2 + H.O.T..\end{aligned}$$

The resulting formula for the gradient becomes:

$$\left.\frac{\partial\phi}{\partial x}\right|_{p_r} = \frac{(\phi_{ll}-\phi_r)h^2 + (\phi_r-\phi_l)(h+h_l)^2}{h(h+h_l)h_l} \tag{2.35}$$

2.3.4 Correction of Gradients

The viscous flux of quantity ϕ under the effect of viscosity ν across face S_{ij} associated with the edge joining point Pi and point Pj is given by

$$\int_{S_{ij}} \nu\vec{\nabla}\phi\cdot\vec{n}dS$$

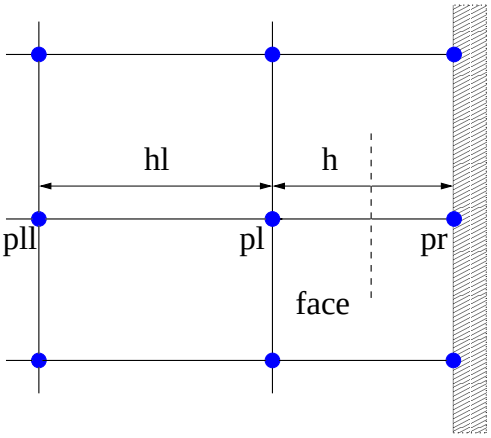


Figure 2.7: Face with one neighboring face on the left side

where $\vec{n}$ is the surface normal vector of unit length of face S_{ij}.
The simple uncorrected version for computation of the viscous fluxes is not used in TAU. The uncorrected formulation reads

$$\int_{S_{ij}} \nu \vec{\nabla}\phi \cdot \vec{n} \mathrm{d}S \;=\; \nu (\vec{\nabla}\phi)|_{\frac{1}{2}} \cdot \vec{n}\, \mathrm{area}(S_{ij}) \;, \qquad (\vec{\nabla}\phi)|_{\frac{1}{2}} \;=\; \frac{1}{2}\left(\vec{\nabla}\phi|_i + \vec{\nabla}\phi|_j \right)$$

Therein, $\mathrm{area}(S_{ij})$ denotes the area of face S_{ij}. The values $\vec{\nabla}\phi|_i$, $\vec{\nabla}\phi|_j$ are the gradients of ϕ in dual grid cell i and j obtained by reconstruction e.g. of Green-Gauss type or least-square type. In the TAU-Code, instead of $(\vec{\nabla}\phi)|_{\frac{1}{2}}$ a correction of the gradient in direction of the edge joining P_i and P_j is applied by using the following formula for $(\vec{\nabla}\phi)|_{\frac{1}{2},\mathrm{corr}}$

$$(\vec{\nabla}\phi)|_{\frac{1}{2},\mathrm{corr}} \;=\; \left(\mathbb{I} - \frac{\Delta\vec{x}_{ij} \otimes \Delta\vec{x}_{ij}}{\Delta\vec{x}_{ij} \cdot \Delta\vec{x}_{ij}} \right) (\vec{\nabla}\phi)|_{\frac{1}{2}} \;+\; \frac{\phi|_i - \phi|_j}{\Delta\vec{x}_{ij} \cdot \Delta\vec{x}_{ij}} \Delta\vec{x}_{ij}$$

with $\Delta\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$ being the difference vector between points P_i and P_j. We use the notation for the dyadic product $(\vec{a} \otimes \vec{b})_{kl} = a_k b_l$ and note that $\mathbb{I} - \frac{\Delta\vec{x}_{ij} \otimes \Delta\vec{x}_{ij}}{\Delta\vec{x}_{ij} \cdot \Delta\vec{x}_{ij}}$ is the projection operator onto the plane normal to $\Delta\vec{x}_{ij}$. From this one can see that $(\vec{\nabla}\phi)|_{\frac{1}{2},\mathrm{corr}}$ is given by $(\vec{\nabla}\phi)|_{\frac{1}{2}}$ in the plane perpendicular to $\Delta\vec{x}_{ij}$ and corrected in direction of $\Delta\vec{x}_{ij}$. In the direction of $\Delta\vec{x}_{ij}$, the gradient is computed using a simple differencing scheme.

2.3.5 Limitation of Gradients

Near shocks the values on the faces have to be limited to avoid overshoots. The limiting is done by an minimum/maximum clipping similar to the strategy presented by Barth and Jesperson [4] with a modification proposed by Venkatakrishnan [72]. If a reconstructed value at any face of the control volume exceeds the minimum (or maximum) of $u(j1)$ and the average values on the faces (given by point $P(j1)$ and the surrounding points $P(j2) \ldots P(jn)$, see Figure 2.3), the gradient $\nabla\vec{u}(j1)$ is scaled by a factor $\Theta(j1)_u$, such that the reconstructed value becomes equal to the averaged minimum (or maximum):

$$\nabla\vec{u}_{lim}(j1) = \nabla\vec{u}(j1) \cdot \Theta(j1)_u \tag{2.36}$$

with

$$\Theta(j1)_u = \frac{(r^u(j1))^2 + 2 \cdot r^u(j1)}{(r^u(j1))^2 + r^u(j1) + 2}. \tag{2.37}$$

The coefficient $r^u(j1)$ has to be determined from the coefficients for the faces of the control volume of $P(j1)$ as:

$$r^u(j1) = \min(2, r^u_{j1,j2}, r^u_{j1,j3}, \ldots, r^u_{j1,jn})$$

where

$$r^u_{j1,jk} = \begin{cases} \frac{u^{min}-u(j1)}{u^r_{j1,jk}-u(j1)} & \text{if} \quad u^r_{j1,jk} < u(j1) \\ \frac{u^{max}-u(j1)}{u^r_{j1,jk}-u(j1)} & \text{if} \quad u^r_{j1,jk} > u(j1) \\ 2 & \text{if} \quad u^r_{j1,jk} = u(j1) \end{cases} . \tag{2.38}$$

Herein u^{max} and u^{min} denote the maximum/minimum of $u(j1)$ and values of u in the neighbor points of $P(j1)$:

$$\begin{aligned} u^{max} &= \max(u(j1), u(j2), u(j3), \ldots, u(jn)), \\ u^{min} &= \min(u(j1), u(j2), u(j3), \ldots, u(jn)), \end{aligned}$$

$u^r_{j1,jk}$ denote the reconstructed (unlimited) value on the control volume face between $P(j1)$ and $P(jk)$:

$$u^r_{j1,jk} = \nabla\vec{u}(j1) \cdot \frac{1}{2} \cdot \left(\vec{P}(jk) - \vec{P}(j1)\right). \tag{2.39}$$

2.4 Central Scheme - Spatial Discretization

The approximation of the governing equations is again done with finite-difference quotients. The explanation follows again with the continuity equation 2.4:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0$$

The shorthand description of above equation reads

$$\rho_t + u\rho_x = 0.$$

If we apply again the appropriate initial and boundary conditions and $u > 0$ this becomes a first order hyperbolic equation. Using a second order difference formula for the discretization of the spatial derivative ρ_x at mesh point i and again a forward difference formula for the time derivative the discrete equation reads:

$$\frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} = -\frac{u}{2\Delta x}\left(\rho_{i+1}^n - \rho_{i-1}^n\right) \tag{2.40}$$

The index n describes the time level. We have now derived a *second-order central* scheme.

Let the points $P(j1)$ and $P(j2)$ be separated by the face $F(i)$: $P(j1) = C_F^P(i,1)$ and $P(j2) = C_F^P(i,2)$, see Figure 2.2. The convective fluxes over the face are computed based on the flow conditions in the points $P(j1)$ and $P(j2)$. The convective fluxes have to be considered as fluxes between the control volumes surrounding the points $P(j1)$ and $P(j2)$. The size and orientation of the face is described by the face normal vector $\vec{F}_i = (F_i^x, F_i^y, F_i^z)^T$ as depicted in Figure 2.2. The central fluxes over the face can then be computed as:

$$\vec{Q}_F^{F,c} = \underbrace{\frac{1}{2}\left(\vec{F}_r(i) + \vec{F}_l(i)\right)}_{convective\ \ terms} - \underbrace{\frac{1}{2}\tilde{\alpha}\left(\vec{w}_r - \vec{w}_l\right)}_{dissipative\ \ terms}. \tag{2.41}$$

$\tilde{\alpha}$ describes the type of the dissipation (see Part 2.4.2), scalar or matrix dissipation. The difference $(\vec{w}_r - \vec{w}_l) = \vec{D}(j1)$ is computed as:

$$\vec{D}(j1) = (\vec{w}_r - \vec{w}_l) = \varepsilon_{(2)}\left(\vec{u}_r - \vec{u}_l\right) - \varepsilon_{(4)}\left(L\left(\vec{u}_r\right) - L\left(\vec{u}_l\right)\right).$$

The vector $\vec{w}$ holds the conservative variables $\rho, \rho u, \rho v, \rho w$ and ρE. The Laplacian (∇^2) of $L(\vec{u}_i)$ is calculated from each neighboring point (more in Part 2.4.3)

$$L\left(\vec{u}_i\right) = \sum_{j=1}^{neighbors}\left(\vec{u}_j - \vec{u}_i\right).$$

As an example the Figure 2.8 shows the amount of points, from point P_{j0} to point P_{j9}, needed for computing the flux over face F_i for a central scheme. Especially the influence of computing the Laplacian (fourth order differences) spreads widely. For point P_{j0} the neighboring points are $P_{j1}, P_{j4}, P_{j5}, P_{j6}, P_{j7}$ and P_{j8}, connected to each other with a green dashed line. For point P_{j1} the neighboring points are $P_{j0}, P_{j8}, P_{j9}, P_{j2}, P_{j3}$ and P_{j4} which are connected to each other with a red dash-dot line.

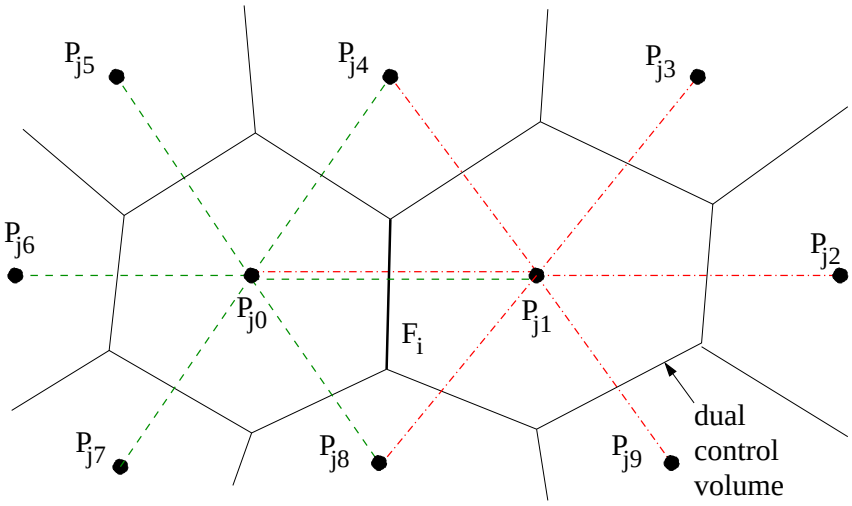


Figure 2.8: Flux computation for a central scheme over face F_i

2.4.1 Spatial Discretization of Convective Terms

The inviscid central fluxes over the face can then be computed as:

$$\vec{Q}_F^{F,c} = F^x(i)\cdot\vec{F}_F^c + F^y(i)\cdot\vec{G}_F^c + F^z(i)\cdot\vec{H}_F^c. \tag{2.42}$$

with

$$\vec{F}_F^c = \frac{1}{2}\begin{pmatrix} (\rho u)(j1)+(\rho u)(j2) \\ (\rho u)(j1)\cdot u(j1)+p(j1)+(\rho u)(j2)\cdot u(j2)+p(j2) \\ (\rho v)(j1)\cdot u(j1)+(\rho v)(j2)\cdot u(j2) \\ (\rho w)(j1)\cdot u(j1)+(\rho w)(j2)\cdot u(j2) \\ (\rho H)(j1)\cdot u(j1)+(\rho H)(j2)\cdot u(j2) \end{pmatrix}$$

$$\vec{G}_F^c = \frac{1}{2}\begin{pmatrix} (\rho v)(j1)+(\rho v)(j2) \\ (\rho u)(j1)\cdot v(j1)+(\rho u)(j2)\cdot v(j2) \\ (\rho v)(j1)\cdot v(j1)+p(j1)+(\rho v)(j2)\cdot v(j2)+p(j2) \\ (\rho w)(j1)\cdot v(j1)+(\rho w)(j2)\cdot v(j2) \\ (\rho H)(j1)\cdot v(j1)+(\rho H)(j2)\cdot v(j2) \end{pmatrix}$$

$$\vec{H}_F^c = \frac{1}{2}\begin{pmatrix} (\rho w)(j1)+(\rho w)(j2) \\ (\rho u)(j1)\cdot w(j1)+(\rho u)(j2)\cdot w(j2) \\ (\rho v)(j1)\cdot w(j1)+(\rho v)(j2)\cdot w(j2) \\ (\rho w)(j1)\cdot w(j1)+p(j1)+(\rho w)(j2)\cdot w(j2)+p(j2) \\ (\rho H)(j1)\cdot w(j1)+(\rho H)(j2)\cdot w(j2) \end{pmatrix}$$

After having determined the fluxes between the control volumes surrounding $P(j1)$ and $P(j2)$ they are added to the fluxes $\vec{Q}_{old}^{F,c}(j1)$ and $\vec{Q}_{old}^{F,c}(j2)$ of the two points that have been computed before. As depicted in Figure 2.2 the normal vector $\vec{F}(i)$ is pointing from the face to the direction of point $P(j2)$. Therefore, the calculated fluxes have to be reversed before they can be added to the already obtained fluxes for point $P(j2)$:

$$\vec{Q}^{F,c}(j1) = \vec{Q}_{old}^{F,c}(j1) + \vec{Q}_F^{F,c}$$

and

$$\vec{Q}^{F,c}(j2) = \vec{Q}_{old}^{F,c}(j2) - \vec{Q}_F^{F,c}.$$

If the convective terms are determined with the central scheme that is described here, a artificial dissipation is required in order to ensure the stability of the computation.

2.4.2 Artificial Dissipation

1. `Scalar Dissipation:` *Scalar_dissipation*
   In this section the determination of an artificial scalar dissipation is described. The dissipation has to be computed for each grid point $P(i)$. The approach follows the strategy described by [47] in order to obtain an adequate scaling of the dissipation for highly stretched cells.
   From equation 2.41 the complete artifical dissipation is

$$(\tilde{\alpha}\ \vec{D})(j1) = \sum_{k=2}^{n} \alpha(j1,jk)\vec{D}(j1,jk)$$

   where $\alpha(j1,jk)$ and $\vec{D}(j1,jk)$ denote the contribution from edge $\overline{P(j1)P(jk)}$.
   Now we consider the contribution to $(\tilde{\alpha}\ \vec{D})(j1)$ coming from the dissipative flux across the dual face F corresponding to the edge connecting $P(j1)$ and $P(j2)$. For the scalar dissipation $\tilde{\alpha}$ becomes the maximum eigenvalue λ_F^c of the flux Jacobian for the face F weighted by a term $\frac{4\phi_F(j1)\phi_F(j2)}{\phi_F(j1)+\phi_F(j2)}$ which accounts for the local grid stretching direction

$$\tilde{\alpha}(j1,j2) = \lambda_F^c \frac{4\phi_F(j1)\phi_F(j2)}{\phi_F(j1)+\phi_F(j2)} \tag{2.43}$$

$$\lambda_F^c = |\vec{v}_F \cdot \vec{F}| + a_F \cdot |\vec{F}| \tag{2.44}$$

   where a_F denotes the speed of sound of face F.
   The artificial dissipation is constructed as a blending of undivided Laplacian and biharmonic operators. The dissipative flux across the dual face F corresponding to the edge connecting $P(j1)$ and $P(j2)$ is given by

$$\vec{D}(j1,j2) = \varepsilon_F^{k(2)} \cdot \left(\vec{W}(j1) - \vec{W}(j2)\right) - \varepsilon_F^{k(4)} \cdot \left(\nabla^2\vec{W}(j1) - \nabla^2\vec{W}(j2)\right). \tag{2.45}$$

   The coefficients $\varepsilon_F^{k(2)}$ and $\varepsilon_F^{k(4)}$ are used to control the amount of dissipation added to the scheme:

$$\varepsilon_F^{k(2)} = k^{(2)} \max(\nu(j1),\nu(j2)) \cdot sc_2, \tag{2.46}$$

$$\nu(j1) = \left|\frac{\nabla^2 p(j1)}{p^{\Sigma}(j1)}\right| = \left|\frac{\sum_{k=2}^{n}(p(jk)-p(j1))}{\sum_{k=2}^{n}(p(jk)+p(j1))}\right| \tag{2.47}$$

$$\nu(j2) = \left|\frac{\nabla^2 p(j2)}{p^{\Sigma}(j2)}\right| = \left|\frac{\sum_{k=1,k\neq 2}^{n}(p(jk)-p(j2))}{\sum_{k=1,k\neq 2}^{n}(p(jk)+p(j2))}\right| \tag{2.48}$$

$$\varepsilon_F^{k(4)} = \max(0, k^{(4)} - \varepsilon_F^{k(2)}) \cdot sc_4. \tag{2.49}$$

   where $\frac{1}{4} \leq k^{(2)} \leq \frac{1}{2}$ and $\frac{1}{64} \leq k^{(4)} \leq \frac{1}{32}$ are user-prescribed constants. The function ν switches between the two dissipation mechanisms. In regions where the solution is smooth, ν and thus $\varepsilon_F^{k(2)}$ are negligible small. Then only the fourth difference term contributes to the artificial dissipation and the scheme is formally third-order accurate on regular meshes. On the other hand, in the vicinity of shocks, ν becomes large and the second-difference term is activated while simultaneously reducing the fourth-difference

term. This is needed to introduce an entropy condition to reduce overshoots near discontinuities and to choose the correct shock relationships. This formulation ensures that the dissipation is dominated by the first order term for regions in the vicinity of shocks, and the scheme is formally only first-order accurate there.
The terms $\phi_F(j1)$, and $\phi_F(j2)$ are introduced in order to take into account the local grid stretching for avoiding excessive numerical dissipation in cases of meshes with high-aspect-ratio cells. The relative size of λ_F^c compared to the total eigenvalue $\lambda^c(j1)$ integrated over the boundary of the entire control volume around point $P(j1)$ can be described by the term $r_F(j1)$ as

$$r_F(j1) = \frac{1}{2} \cdot \max\left[0\,;\ \frac{\lambda^c(j1) - \lambda_F^c}{\lambda_F^c} \right],$$

with

$$\lambda^c(j1) = \sum_{k=2}^{n} |\vec{v}_F \cdot \vec{F}| + a_F \cdot |\vec{F}|.$$

The corresponding terms $\lambda^c(j2)$ and $r_F(j2)$ are defined analogously. From this the terms $\phi_F(j1)$ and $\phi_F(j2)$ can be calculated as

$$\phi_F(j1) = r_F(j1)^{\omega}\,, \qquad \phi_F(j2) = r_F(j2)^{\omega}$$

with weighting exponent $\omega = 0.5$.
The scaling factors sc_2 and sc_4 are introduced in order to avoid a dependency of the dissipation on the number of neighbors. The size of the factors is chosen such that for six neighbors the dissipation equals the dissipation in a structured scheme:

$$sc_2 = \frac{3}{n_P^P(j1)} + \frac{3}{n_P^P(j2)},$$

and

$$sc_4 = \frac{9}{n_P^P(j1) \cdot (1 + n_P^P(j1))} + \frac{9}{n_P^P(j2) \cdot (1 + n_P^P(j2))}.$$

From the boundary faces additional contributions have to be considered for the boundary points. Consider a boundary face $F_b(i)$ of the boundary part b with a point $P(j1) = C_{F_b}^P(b,i)$ related to the face. As shown in Figure 2.9 the vector of flow variables $\vec{W}$ can be extrapolated over the boundary, employing the quantities in a suited field point $P(j3) = C_{F_b}^{P_r}(b,i)$ and the quantities in the boundary point $P(j1)$. The extrapolated values $\vec{W}^m(j1)$ for the mirrored point $P^m(j1)$ are given by

$$\vec{W}^m(j1) = 2 \cdot \vec{W}(j1) - \vec{W}(j3)$$

The point $P(j3)$ is the point lying most normal in the flow field to the boundary face, see also Part 1.5.1. The laplacians for $P^m(j1)$ equal the laplacians in the origin $P(j1)$, hence there is only a contribution for the second order dissipation coming from the boundary face. The dissipative fluxes are determined for all boundary faces. All values required to compute the dissipation are calculated as described above, with the exception that no averaging is needed.
For coarser grid levels (see also Part 2.5.7) the dissipation is computed in a more simple way by considering the second order dissipation only. The coefficient $\varepsilon_F^{k(2)}$ in this case does not depend on the laplacian of pressure but only on the number of neighbors:

$$\varepsilon_F^{k(2)} = k_{coarse}^{(2)} \cdot sc_2,$$

with $k_{coarse}^{(2)} = 0.15$.

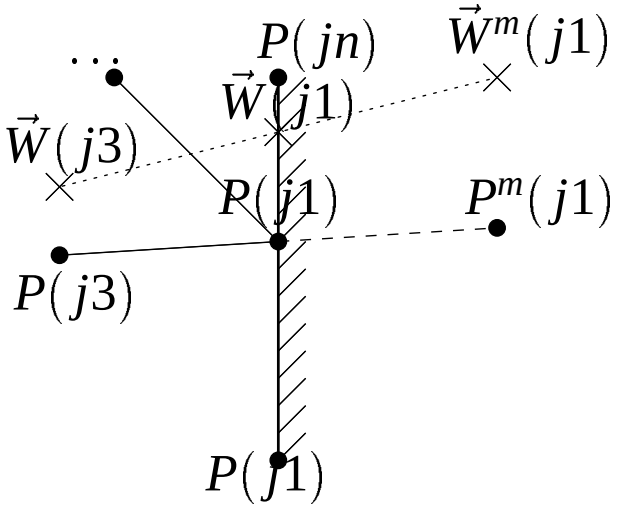


Figure 2.9: Extrapolation of flow quantities over the boundary

2. `Matrix Dissipation`: *Matrix_dissipation*
In this section the determination of an artificial matrix dissipation is described. The dissipation has to be computed for each grid point $P(i)$. The approach follows the strategy described by Turkel [23]. This gives an appropriate viscosity for each wave component.

From equation 2.41 the complete artifical dissipation is

$$\tilde{\alpha}\ \vec{D}(j1).$$

In order to imitate the upwind Roe [59] type algorithm we now replace the scalars in equation 2.44 by matrices and $\tilde{\alpha}$ becomes:

$$\begin{aligned}
\tilde{\alpha}(\vec{u}_l,\vec{u}_r,\vec{n}_{lr},\lambda) &= T|\Lambda|T^{-1} \\
\text{with} \quad |\Lambda| &= \begin{pmatrix} |\lambda_1| & 0 & 0 & 0 & 0 \\ 0 & |\lambda_2| & 0 & 0 & 0 \\ 0 & 0 & |\lambda_3| & 0 & 0 \\ 0 & 0 & 0 & |\lambda_3| & 0 \\ 0 & 0 & 0 & 0 & |\lambda_3| \end{pmatrix} \\
\text{with the entries} \quad \lambda_1 &= v_n\|\vec{F}\| + a_F\|\vec{F}\| \\
\lambda_2 &= v_n\|\vec{F}\| - a_F\|\vec{F}\| \\
\lambda_3 &= v_n\|\vec{F}\| \\
\text{with} \quad v_n &= v_x\cdot n_x + v_y\cdot n_y + v_z\cdot n_z
\end{aligned}$$

a_F denotes the speed of sound of face F. The velocity v_n is the normal component of the face velocity vector. Near stagnation points where λ_3 can get near zero or at shocks where λ_2 gets close to zero this formulation can lead to difficulties. To prevent the entries becoming zero the factor δ is introduced. δ switches between scalar dissipation $\delta = 1$ and matrix dissipation $\delta = 0$. The entries in the matrix $|\Lambda|$ are now described as:

$$\begin{aligned}
|\lambda_1| &= max(|V_n + A_F|), \delta(|V_n| + A_F) \\
|\lambda_2| &= max(|V_n - A_F|), \delta(|V_n| + A_F) \\
|\lambda_3| &= max(|V_n|), \delta(|V_n| + A_F)
\end{aligned}$$

The normal velocity vector $V_n = v_n\|\vec{F}\|$ and the sound of speed for face F is $A_F = a_F\|\vec{F}\|$.

The dissipative fluxes for a control volume surrounding point $P(j1)$ with $n-1$ neighboring points $P(j2)\dots P(jn)$ can now be computed as for scalar dissipation:

$$\vec{D}(j1) = \sum_{k=2}^{n} \left\{ \left[\varepsilon_F^{k(2)} \cdot \left(\vec{W}(j1) - \vec{W}(jk) \right) - \right. \right. \\ \left. \left. \varepsilon_F^{k(4)} \cdot \left(\nabla^2 \vec{W}(j1) - \nabla^2 \vec{W}(jk) \right) \right] \cdot \frac{A_F(j1) + A_F(jk)}{2} \right\}.$$

The computation of the above equation is performed equivalent to the scalar dissipation model.

2.4.3 Determination of Laplacians

The contributions to the Laplacians of the conservative variables $\nabla^2 \vec{W}(j1)$ with $\{\nabla^2 \vec{W}(m) \,|\, m = 1,\dots N^P\}$ and the pressure $\nabla^2 p(j1)$ with $\{\nabla^2 p(m) \,|\, m = 1,\dots N^P\}$ and also the face sum of the pressure $p^\Sigma(j1)$ with $\{p^\Sigma(m) \,|\, m = 1,\dots N^P\}$ are calculated for each point $P(j1)$ related to the face F_i that is separating point $P(j1)$ and point $P(j2)$ as:

$$\begin{aligned} \Delta_F \left(\nabla^2 \vec{W}(j1) \right) &= \vec{W}(j2) - \vec{W}(j1) \\ \Delta_F \left(\nabla^2 p(j1) \right) &= p(j2) - p(j1) \\ \Delta_F \left(p^\Sigma(j1) \right) &= p(j1) + p(j2). \end{aligned}$$

The contributions are added to the values $\nabla^2 \vec{W}(j1)_{old}$, $\nabla^2 \vec{p}(j1)_{old}$ and $p^\Sigma(j1)_{old}$ that have been computed before:

$$\begin{aligned} \nabla^2 \vec{W}(j1) &= \nabla^2 \vec{W}(j1)_{old} + \Delta_F \left(\nabla^2 \vec{W}(j1) \right) \\ \nabla^2 p(j1) &= \nabla^2 p(j1)_{old} + \Delta_F \left(\nabla^2 p(j1) \right) \\ p^\Sigma(j1) &= p^\Sigma(j1)_{old} + \Delta_F \left(p^\Sigma(j1) \right). \end{aligned}$$

Furthermore, contributions from the boundary faces have to be added. Consider a boundary face $F_b(i)$ of the boundary part b with a point $P(j1) = C_{F_b}^P(b,i)$ related to the face. For point $P(j1)$ a neighboring point $P(j3) = C_{F_b}^{P_r}(b,i)$ is determined that lies normal to the boundary face, see Part 1.5.1. As the flow quantities are extrapolated linearly over the boundary (see Figure 2.9), the contributions to the laplacians and the pressure sum for point $P(j1)$ are given by:

$$\begin{aligned} \Delta_{F_b} \left(\nabla^2 \vec{W}(j1) \right) &= \vec{W}(j1) - \vec{W}(j2) \\ \Delta_{F_b} \left(\nabla^2 \vec{p}(j1) \right) &= p(j1) - p(j2) \\ \Delta_{F_b} \left(p^\Sigma(j1) \right) &= 3 \cdot p(j1) - p(j2). \end{aligned}$$

These contributions have also to be added to the values $\nabla^2 \vec{W}(j1)_{old}$, $\nabla^2 \vec{p}(j1)_{old}$ and $p^\Sigma(j1)_{old}$ that have been computed before. After the contributions are determined for all grid faces and summed up, the laplacian of pressure $\nabla p(j1)$ is divided by the face sum of pressure $p^\Sigma(j1)$ and $\nu(j1)$ is computed for each point $P(j1)$:

$$\nu(j1) = \frac{\nabla^2 \vec{p}(j1)}{p^\Sigma(j1)}.$$

2.5 Solution technique for steady state problems

The temporal variation of the flow quantities can be written in general form for a point $P(j1)$ as:

$$\frac{d}{dt}\vec{W}(j1)+\vec{R}(j1)=0. \tag{2.50}$$

A comparison with equation 2.8 gives for the residual $\vec{R}(j1)$:

$$\vec{R}(j1)=\frac{1}{V(j1)}\cdot\vec{Q}^F(j1). \tag{2.51}$$

The integration in time is performed utilizing an explicit Runge-Kutta scheme, as described by Jameson [33]:

$$\begin{array}{lcl}\vec{W}^{(0)}(j1) & = & \vec{W}(j1)(n)\\ \vec{W}^{(1)}(j1) & = & \vec{W}(j1)^{(0)}-\alpha(j)\Delta t(j1)\vec{R}_v^{(0)}(j1)\\ & \vdots & \\ \vec{W}^{(a)}(j1) & = & \vec{W}(j1)^{(0)}-\alpha_a\Delta t(j1)\vec{R}_v^{(a-1)}(j1)\\ \vec{W}(j1)(n+1) & = & \vec{W}^{(a)}(j1)\end{array} \tag{2.52}$$

with

$$\vec{R}_v(j1)=V(j1)\cdot\vec{R}(j1).$$

In this equation the residual $\vec{R}(j1)$ equals the fluxes $\vec{Q}(j1)^F$ over the control volume boundaries. For multigrid calculations the residual $\vec{R}(j1)$ is the sum of the fluxes $\vec{Q}^F(j1)$ over the boundaries of the control volume and the forcing term $\vec{Q}^P(j1)$ coming from the next finer grid. Details concerning the multigrid algorithm can be found in Part 2.5.7.

2.5.1 Steady state problems

First we consider the so-called *steady state* case, in which a *time-independent solution* exists. From the steady state condition $\frac{d\vec{W}}{dt}=0$, Eq. (2.50) becomes

$$\vec{R}(j1)=0. \tag{2.53}$$

This problem is solved by introducing the corresponding time-dependent problem with fictitious pseudo-time t^* and seeking its steady-state solution

$$\frac{d}{dt^*}\vec{W}(j1)+\vec{R}(j1)=0. \tag{2.54}$$

The discretization with respect to the fictitious pseudo-time is performed using the low-storage K-step Runge-Kutta scheme given in Sec. 2.5, see equation 2.52, where $\Delta t^*(j1)$ denotes the pseudo-time step width:

$$\begin{array}{lcl}\vec{W}^{(0)}(j1) & = & \vec{W}(j1)(n)\\ \vec{W}^{(1)}(j1) & = & \vec{W}(j1)^{(0)}-\alpha(j)\Delta t^*(j1)\vec{R}_v^{(0)}(j1)\\ & \vdots & \\ \vec{W}^{(a)}(j1) & = & \vec{W}(j1)^{(0)}-\alpha_a\Delta t^*(j1)\vec{R}_v^{(a-1)}(j1)\\ \vec{W}(j1)(n+1) & = & \vec{W}^{(a)}(j1)\end{array} \tag{2.55}$$

with

$$\vec{R}_v(j1)=V(j1)\cdot\vec{R}(j1).$$

In this equation the residual $\vec{R}(j1)$ equals the fluxes $\vec{Q}(j1)^F$ over the control volume boundaries. For multigrid calculations the residual $\vec{R}(j1)$ is the sum of the fluxes $\vec{Q}^F(j1)$ over the boundaries of the control volume and the forcing term $\vec{Q}^P(j1)$ coming from the next finer grid. Details concerning the multigrid algorithm can be found in Part 2.5.7.

2.5.2 Acceleration Techniques

Various techniques are known in the literature, Jameson [32], Brandt [6], Kroll/Jain [35], to accelerate the convergence of the solution to steady state. The TAU-Code has implemented the following techniques:

- Local Time Stepping
- Residual Smoothing
  - Explicit Residual Smoothing
  - Implicit Residual Smoothing
- Multigrid

The local time stepping is based on a modification of the differential equations whereas the residual smoothing is an improvement during the solution process. Further reducement of computational effort is achieved with multiple grid techniques such as multigrid.

2.5.3 Local Time Stepping

The idea of local time stepping is to allow that in each control volume the solver uses the maximum allowed time step. The time stepping operates at its stability limit everywhere in the flowfield.
To be more precise, we consider again the fictitious pseudo-time problem (with pseudo time t^*)

$$\frac{d}{dt^*}\vec{W}(j1)+\vec{R}(j1)=0. \tag{2.56}$$

The question is how to choose the pseudo time step width $\Delta t^*(j1)$ for cell $j1$ in the corresponding discrete formulation. For dual cell $j1$, let $\Delta t(j1)$ denote the local time step width, see Section 2.5.15 for its computation.
In *global time stepping*, we set $\Delta t^*(j1)=\min_{j=1}^{N_c}\{\ \Delta t(j)\ \}$, i.e. the pseudo time step width $\Delta t^*(j1)$ is given globally by the minimum time step over all N_c dual grid cells (here N_c denotes the number of dual grid cells). Global time stepping is needed to compute a time-accurate solution.
For steady state problems the solution of (2.56) remains unchanged if we replace d/dt^* by $(1/A)d/dt^*$ with local acceleration parameter A:

$$\frac{1}{A(j1)}\frac{d}{dt^*}\vec{W}(j1)+\vec{R}(j1)=0. \tag{2.57}$$

In *local time-stepping* we choose $\Delta t^*(j1)\equiv 1$ for all dual cells $j1$ and $A(j1)=\Delta t(j1)$ where $\Delta t(j1)$ denotes the local time step width. Obviously, this technique can be used only for time integration of steady state flows.

2.5.4 Residual Smoothing

After having calculated the changes $\vec{R}_t(j1)$ with

$$\vec{R}_t(j1) = \vec{R}_v(j1) \cdot \alpha(j) \Delta t(j1),$$

of the flow conditions in point $P(j1)$ these changes can be smoothed explicitly or implicitly in order to increase the robustness of the scheme. In the following, the changes $\vec{R}_t(j1)$ are also called residuals.

2.5.5 Explicit Residual Smoothing

The residuals can be smoothed explicitly employing a Laplacian type smoother. The residuals in a point $P(j1)$ with the neighboring points $P(jk) \,|\, k = 2, \ldots n\}$ can be smoothed in an iterative process. In the mth iteration the residuals ${}_{g_l}\vec{C}(i1)$ are given by:

$${}^{m}\vec{R}_t(j1) = (1-\omega) \cdot {}^{m-1}\vec{R}_t(j1) + \frac{\omega}{n-1} \sum_{k=2}^{n} \left({}^{m-1}\vec{R}_t(jk) \right). \tag{2.58}$$

For very anisotropic grids showing large differences in the face sizes the surface of a control volume for a point $P(j1)$ is composed of, a weighting of the contributions with respect to the face size is used. Equation 2.58 written with such a weighting reads:

$${}^{m}\vec{R}_t(i) \;=\; (1-\omega) \cdot {}^{m-1}\vec{R}_t(i) + \frac{\omega}{\sum_{k=2}^{m} |\vec{F}(i1,ik)|} \sum_{k=2}^{n} \left({}^{m-1}\vec{R}_t(jk) \cdot |\vec{F}(i1,ik)| \right). \tag{2.59}$$

The changes in the flow quantities which are caused by the addition of the residuals may lead to a violation of the boundary conditions. The initialization of a velocity on a viscous wall e.g. is not allowed. Therefore, the residuals in the boundary points have to be adapted to the respective boundary conditions. This method equals the approach as described for the treatment of the fluxes in the boundary point, alone that instead of the fluxes the residuals have to be modified.

2.5.6 Implicit Residual Smoothing

Alternatively, the residuals can be smoothed implicitly solving the implicit equation

$$\tilde{\vec{R}}_t(j1) = \vec{R}_t(j1) + \sum_{k=2}^{n} \left[\varepsilon_F \cdot (\tilde{\vec{R}}_t(jk) - \tilde{\vec{R}}_t(j1) \right] \tag{2.60}$$

where $\tilde{\vec{R}}_t(j1)$ is the smoothed residual. The equation 2.60 is solved iteratively employing Jacobi iterations. The smoothed residual in the mth iteration is given by:

$${}^{m}\tilde{\vec{R}}_t(j1) = \frac{1}{\sum_{k=2}^{n} \varepsilon_F} \cdot \left(\vec{R}_t(j1) + \sum_{k=2}^{n} \left[\varepsilon_F \cdot {}^{m-1}\tilde{\vec{R}}_t(jk) \right] \right). \tag{2.61}$$

The scaling factor ε_F is computed with reference to Martinelli [45] in dependence of the ratio between the implicit CFL-number and the largest allowable unsmoothed CFL-number CFL^{ex}:

$$\varepsilon_F = \max \left(0, \frac{1}{4} \left\{ \left[\frac{CFL}{CFL^{ex}} \cdot \frac{2\lambda_F^c}{\lambda_F^c + \lambda^c(j1)} \cdot (1 + r_F(j1)^{\omega}) \right]^2 - 1 \right\} \right). \tag{2.62}$$

Herein, λ_F^c denotes the maximum eigenvalue of the flux Jacobian for the face F:

$$\lambda_F^c = |\vec{v}_F \cdot \vec{F}| + a_F \cdot |\vec{F}|.$$

The term $r_F(j1)$ reflects the relative size of λ_F^c compared to the total eigenvalue $\lambda^c(j1)$ integrated over the boundary of the entire control volume around point $P(j1)$:

$$r_F(j1) = \frac{1}{2} \cdot \frac{\lambda^c(j1) - \lambda_F^c}{\lambda_F^c},$$

where

$$\lambda^c(j1) = \sum_{k=2}^{n} |\vec{v}_F \cdot \vec{F}| + a_F \cdot |\vec{F}|.$$

2.5.7 Multigrid

The construction of the grids by agglomerating control volumes during the preprocessing is described in Part 1.7. The first section of this chapter gives a brief description of the multigrid strategy. The following sections deal with the transfer operators between the different grid levels needed to extend the single grid algorithm introduced in Part 2. These operators can be divided into three groups:

- *Restriction operators*
  The transfer from a finer to a coarser grid is called restriction. During the multigrid algorithm the solution and residuals of the equations are restricted to the next coarser grid level.
- *Prolongation operators*
  The transfer from a coarser to a finer grid is called prolongation. During the multigrid algorithm the corrections of the solution are prolongated to the next finer grid level.
- *Smoothing operators*
  The equations solved on the coarser grid levels are only valid for the low frequency error of the solution iterated on the finer mesh. Therefore the residuals as well as the corrections are smoothed during the multigrid algorithm to improve the convergence.

2.5.8 General Idea of Multigrid

The multigrid method does not depend on the equations to be solved. Hence, it will be explained by considering a scalar non-linear equation:

$$L(w) = S$$

with the solution w and a solution independent right hand side S. On the finest grid f the equation is discretized as a system of equations:

$$L_f(w_f) = S_f$$

which has to be solved simultaneously for all control volumes. The true solution w_f^{true} of the discretized equation is iterated numerically. Each iteration step starts with a solution w_f^0 and ends with w_f^1. In the code a Runge-Kutta scheme, described in Part 2.5, is used as iteration method. The normal convergence behavior of an iteration method slows down after a rapid start,

due to the fact, that low frequency errors are hardly dampened. The advantage of a multigrid method is, that these low frequency errors (attached to a long wavelength) can be well resolved on a coarser mesh. On a coarser mesh the calculation of these errors is much easier because its wavelength becomes shorter compared to the grid spacing and the number of equations to be solved decreases with the number of agglomerated volumes. In addition the higher order terms of an upwind scheme can be neglected, because they are not necessary to resolve the dampening of a smooth low frequency error. But the equations on the coarser grid have to be changed in a way, that the solution converges to the fine grid solution. Therefore, an iteration step on the fine grid results in a residual *res*:

$$res_f = L_f(w_f^1) - S_f.$$

Subtracting from the true solution provides:

$$L_f(w_f^{true}) - L_f(w_f^1) = -res_f.$$

For the smooth low frequency errors this leads on a coarser mesh (m) to:

$$L_m(w_m^{true}) = I_{fm}^r(-res_f) + L_m(w_m^0). \tag{2.63}$$

The restriction operator I^r of the residual has to obey:

$$I_{fm}^r(-res_f) = -I_{fm}^r(res_f) \Rightarrow I_{fm}^r(0) = 0.$$

If the residual on the fine grid equals zero, then the starting solution is a true solution of equation 2.63. As a starting value for the iteration on the mesh m a restriction I^w of the iterated fine grid solution is used:

$$w_m^0 = I_{fm}^w(w_f^1).$$

The difference between the best approximation (w^{ba}) of the true solution and the starting solution is called correction C:

$$C_m = w_m^{ba} - w_m^0 = w_m^{ba} - I_{fm}^w(w_f^1).$$

This correction is prolongated to the finer grid to improve the solution there:

$$w_f^{ba} = w_f^1 + I_{mf}^c(C_m).$$

The same approach can be applied to calculate the best approximation to the true solution of equation 2.63 on the grid m by considering an even coarser mesh c. An iteration step on grid m stops with a residual:

$$res_m = L_m(w_m^1) - L_m(w_m^0) - I_{fm}^r(-res_f),$$

which can be used to formulate equation 2.63 on the next coarser grid (c):

$$L_c(w_c^{true}) = I_{mc}^r(-res_m) + L_c(w_c^0).$$

The way to a coarser mesh can be taken again and again, but in practice the use of more than about four meshes does not increase the convergence sufficiently to pay for the additional operations. On the coarsest grid the iterated solution is taken as the best approximation.

Due to the different equations on the fine grid and on the coarser ones the residuals *res* are defined differently depending on the grid level. In a different notation these residuals are called defects and the name residual is then used only for the residual R of the governing equation in the sense of equation 2.51. The equation on the coarser level 2.63 then reads as:

$$\frac{d}{dt}w_m + R_m - F_m = 0,$$

with a forcing function F_m given by:

$$F_m = R_m(w_m^0) - I_{fm}^r\left(R_f(w_f^1) - F_f\right).$$

The forcing function on the finest grid is equal to zero.

2.5.9 Motivation: The Correction Scheme for linear problems

As a motivation for the multigrid scheme for nonlinear problems used in the TAU-Code (called full approximation scheme, FAS), we start with the well-known multigrid scheme for linear problems referred to as correction scheme, see [10] pp.13 and pp.81. For this purpose, we consider the *linear* problem

$$L\,u \;=\; f \tag{2.64}$$

We introduce the following notation: If we solve (2.64) on grid level k we write $L^k\,u^k \;=\; f^k$. For the Correction Scheme, on the finest grid level 1 we seek u^1 s.t.

$$L^1 u^1 \;=\; f^1$$

On each coarse grid level $k > 1$ and given an approximation $\tilde{u}^{k-1}$ from the next finer level, we seek the correction c^k s.t.

$$L^k(I_{k-1}^k \tilde{u}^{k-1} + c^k) \;=\; f^k \quad \Leftrightarrow \quad L^k\,c^k \;=\; r^k \;\equiv\; I_{k-1}^k(f^{k-1} - L^{k-1}\tilde{u}^{k-1})$$

where we use that L is *linear*. Once this equation has been solved, c^k is used as a correction to obtain a new fine-grid solution

$$u^{k-1} \;=\; \tilde{u}^{k-1} + I_k^{k-1} c^k$$

If we perform N relaxation smoothing steps on level k, starting with u_0^k and with right hand side term being f^k, then we write $J_N(u_0^k, f^k)$. Moreover we write $u_N^k = J_N(u_0^k, f^k)$ for the smoothed solution. Additionally, we introduce restriction and prolongation operators. The restriction operator I_k^{k+1} restricts a solution on level k to the next coarser level $k+1$. The prolongation operator I_{k+1}^k prolongates a solution from level $k+1$ to the next finer level k.
The correction scheme then reads (see [10], pp.13)

INPUT: u_0^k

OUTPUT: $MGC(k, u_0^k, f^k)$ is defined by the recursion

IF $k = I_C$ **THEN**

(1) Determine (exact) solution c^k of $L^k c^k = f^k$

(2) **RETURN** c^k

ELSE

(1) Initial guess : $u_0^k = 0$

(2) Perform N_1 relaxation/smoothing steps : $u_a^k = J_{N_1}(u_0^k, f^k)$

(3) DO Recursion : Compute correction on next coarser grid
$c^{k+1} = MGC(k+1, u_0^{k+1}, I_k^{k+1}(f^k - L^k u_a^k))$.

(4) Correction : $u_b^k = u_a^k + I_{k+1}^k c^{k+1}$

(5) Perform N_2 relaxation/smoothing steps : $u_c^k = J_{N_2}(u_b^k, f^k)$

(6) **RETURN** u_c^k

As an example we study explicitely write down the above scheme for $I_C = 3$

COMPUTE $MGC(1, u_0^1, f^1)$

(1.1) Initial guess : $u_0^1 = 0$

(1.2) Perform N_1 relaxation/smoothing steps : $u_a^1 = J_{N_1}(u_0^1, f^1)$

(1.3) DO Recursion : Compute correction on next coarser grid
$c^2 = MGC(2, u_0^2, I_1^2(f^1 - L^1 u_a^1))$.

(2.1) Initial guess : $u_0^2 = 0$

(2.2) Perform N_1 relaxation/smoothing steps : $u_a^2 = J_{N_1}(u_0^2, f^2)$

(2.3) DO Recursion : Compute correction on next coarser grid
$c^3 = MGC(3, u_0^3, I_2^3(f^2 - L^2 u_a^2))$.

(3.1) Solve $L^3 c^3 = I_2^3(f^2 - L^2 u_a^2)$

(3.2) **RETURN** c^3 .

(2.4) Correction : $u_b^2 = u_a^2 + I_3^2 c^3$

(2.5) Perform N_2 relaxation/smoothing steps : $u_c^2 = J_{N_2}(u_b^2, f^2)$

(2.6) **RETURN** $c^2 = u_c^2$

(1.4) Correction : $u_b^1 = u_a^1 + I_2^1 c^2$

(1.5) Perform N_2 relaxation/smoothing steps : $u_c^1 = J_{N_2}(u_b^1, f^1)$

(1.6) **RETURN** $u_c^1 = MGC(1, u_0^1, f^1)$

2.5.10 The Full Approximation Scheme

For the Correction Scheme, on the finest grid level 1 we seek u^1 s.t.

$$L^1 u^1 \;=\; f^1$$

On each coarse grid level $k > 1$ and given an approximation $\tilde{u}^{k-1}$ from the finer level, we seek c^k s.t.

$$L^k(I_{k-1}^k \tilde{u}^{k-1} + c^k) \;=\; f^k \quad \Leftrightarrow \quad L^k\, c^k \;=\; r^k \;\equiv\; I_{k-1}^k(f^{k-1} - L^{k-1}\tilde{u}^{k-1})$$

where we use that L is *linear*. Once this equation has been solved, c^k is used as a correction to obtain a new fine-grid solution

$$u^{k-1} \;=\; \tilde{u}^{k-1} + I_k^{k-1} c^k$$

On the other hand, in the Full Approximation Scheme (see [10] pp.81) we perform as follows: On the finest grid level 1 we seek u^1 s.t.

$$L^1 u^1 \;=\; f^1$$

On each coarse grid level $k > 1$ and given an approximation $\tilde{u}^{k-1}$ from the finer level, we seek $\hat{u}^k = I_{k-1}^k \tilde{u}^{k-1} + c^k$ s.t.

$$L^k \hat{u}^k \;=\; \hat{f}^k \;\equiv\; L^k(I_{k-1}^k \tilde{u}^{k-1}) + I_{k-1}^k(f^{k-1} - L^{k-1}\tilde{u}^{k-1}) \tag{2.65}$$

where we do not need that L is linear. Once this equation has been solved, the new fine-grid solution is obtained as

$$u^{k-1} \;=\; \tilde{u}^{k-1} + I_k^{k-1}(\hat{u}^k - I_{k-1}^k \tilde{u}^{k-1})$$

It is worthwhile mentioning that the full approximation scheme reduces to the correction scheme if L is a linear operator. To see this, we rearrange (2.65) as

$$\begin{aligned} L^k\hat{u}^k \;-\; L^k(I_{k-1}^k\tilde{u}^{k-1}) \;&=\; I_{k-1}^k(f^{k-1} - L^{k-1}\tilde{u}^{k-1}) \\ \Leftrightarrow \qquad L^k c^k \;&=\; I_{k-1}^k(f^{k-1} - L^{k-1}\tilde{u}^{k-1})\,, \qquad\qquad c^k \;\equiv\; \hat{u}^k - I_{k-1}^k\tilde{u}^{k-1}\,. \end{aligned}$$

given that L is linear. Moreover, using this notation the fine grid updated solution becomes

$$u^{k-1} \;=\; \tilde{u}^{k-1} + I_k^{k-1} c^k\,.$$

Thus the recursion form of the Full Approximation Scheme reads as follows:

INPUT: u_0^k

OUTPUT: $MGC(k, u_0^k, f^k)$ is defined by the following recursion

IF $k = I_C$ **THEN**

(1) Determine exact solution $\hat{u}^k$ of $L^k\hat{u}^k = f^k$

(2) **RETURN** $\hat{u}^k$

ELSE

(1) Initial guess : $u_0^k = 0$

(2) Perform N_1 relaxation/smoothing steps : $u_a^k = J_{N_1}(u_0^k, f^k)$

(3) DO Recursion : Compute solution $\hat{u}^{k+1}$ on next coarser grid
$\hat{u}^{k+1} = MGC(k+1, u_0^{k+1}, L^{k+1}(I_k^{k+1}u_a^k) + I_k^{k+1}(f^k - L^k u_a^k))$.

(4) Correction : $u_b^k = u_a^k + I_{k+1}^k(\hat{u}^{k+1} - I_k^{k+1}u_a^k)$

(5) Perform N_2 relaxation/smoothing steps : $u_c^k = J_{N_2}(u_b^k, f^k)$

(6) **RETURN** u_c^k

Again, as an example we study explicitely write down the above scheme for $I_C = 3$

COMPUTE $MGC(1, u_0^1, f^1)$

(1.1) Initial guess : $u_0^1 = 0$

(1.2) Perform N_1 relaxation/smoothing steps : $u_a^1 = J_{N_1}(u_0^1, f^1)$

(1.3) DO Recursion : Compute solution on next coarser grid
$\hat{u}^2 = MGC(2, u_0^2, L^2(I_1^2 u_a^1) + I_1^2(f^1 - L^1 u_a^1))$.

(2.1) Initial guess : $u_0^2 = 0$

(2.2) Perform N_1 relaxation/smoothing steps : $u_a^2 = J_{N_1}(u_0^2, f^2)$

(2.3) DO Recursion : Compute solution on next coarser grid
$\hat{u}^3 = MGC(3, u_0^3, L^3(I_2^3 u_a^2) + I_2^3(f^2 - L^2 u_a^2))$.

(3.1) Solve $L^3\hat{u}^3 = L^3(I_2^3 u_a^2) + I_2^3(f^2 - L^2 u_a^2)$

(3.2) **RETURN** $\hat{u}^3$

(2.4) Correction : $u_b^2 = u_a^2 + I_3^2(\hat{u}^3 - I_2^3 u_a^2)$

(2.5) Perform N_2 relaxation/smoothing steps : $u_c^2 = J_{N_2}(u_b^2, f^2)$

(2.6) **RETURN** $\hat{u}^2 = u_c^2$

(1.4) Correction : $u_b^1 = u_a^1 + I_2^1(\hat{u}^2 - I_1^2 u_a^1)$

(1.5) Perform N_2 relaxation/smoothing steps : $u_c^1 = J_{N_2}(u_b^1, f^1)$

(1.6) **RETURN** u_c^1

As an important notion, let us point out that (2.65) can be rewritten as

$$L^k\hat{u}^k = f^k + \tau_{k-1}^k\,, \qquad \tau_{k-1}^k \equiv L^k(I_{k-1}^k\tilde{u}^{k-1}) - I_{k-1}^k(L^{k-1}\tilde{u}^{k-1}) \tag{2.66}$$

with $f^k = I_{k-1}^k(f^{k-1})$. The additional right hand side term τ_{k-1}^k (w.r.t. the original coarse grid equation) is often referred to as *fine-to-coarse defect correction*.
A major advantage of FAS over CS is that it is directly applicable to nonlinear problems.

2.5.11 The Full Approximation Scheme for the compressible NSE

The Full Approximation Scheme can be directly applied to the compressible Navier-Stokes equations. We consider the non-linear problem

$$L(u) \;=\; f \tag{2.67}$$

In the case of the steady-state compressible Navier-Stokes equations we have

$$L(u) \;\equiv\; \vec{\mathcal{R}}(\vec{u})\,, \qquad f \;\equiv\; 0$$

and in the unsteady case we have (cf. (2.77))

$$L(u) \;\equiv\; \vec{\mathcal{R}}_B^{DTS}(\vec{u})\,, \qquad f \;\equiv\; 0$$

However, it should be kept in mind that for $k = 1$ (i.e., on the finest grid) $f^1 = 0$ but f^k $(k > 1)$ is not zero due to the fine-to-coarse defect correction.

Finally, we have to specify the relaxation/smoothing operation $J_N(u_0^k, f^k)$ and the solution operation on the coarsest grid level I_C.
In principle, each iterative solver of the nonlinear problem (2.67) can be used as a smoother for a multigrid scheme, if it damps the high frequency components of the error on the corresponding grid level. In the TAU-Code multigrid implementation, smoothing is performed by applying an explicit K-stage low storage Runge-Kutta scheme as iterative solver to the pseudo-instationary problem corresponding to (2.67)

$$\frac{du}{dt^*} + L(u) \;=\; f$$

with pseudo time t^*. Only one smoothing step before the recursion is performed, i.e. $N_1 = 1$, and moreover $N_2 = 0$ is used. Thus $J_1(u_0^k, f^k)$ is defined as follows:

Smoothing/ relaxation $J_1(u_0^k, f^k)$

INPUT: u_0^k

OUTPUT: $J_1(u_0^k, f^k)$

FOR $j = 1, \ldots, K$ **DO** $u_j^k = u_0^k - \alpha_j \Delta t^* (L(u_{j-1}^k) - f^k)$

RETURN $J_1(u_0^k, f) = u_K^k$

To this end, on the coarsest grid level $k = I_C$, we define $MGC(I_C, u_0^{I_C}, f^{I_C})$ also by applying one step of an explicit K-stage low storage Runge-Kutta scheme, i.e.

Solution on coarsest grid level $MGC(I_C, u_0^{I_C}, f^{I_C})$

INPUT: $u_0^{I_C}$

OUTPUT: $MGC(I_C, u_0^{I_C}, f^{I_C})$

FOR $j = 1, \ldots, K$ **DO** $u_j^{I_C} = u_0^{I_C} - \alpha_j \Delta t^* (L(u_{j-1}^{I_C}) - f^{I_C})$

RETURN $MGC(I_C, u_0^{I_C}, f^{I_C}) = u_K^{I_C}$

2.5.12 Multigrid restrictions

The conservative variables as well as the residuals or forcing functions have to be restricted from the finer grid to the next coarser grid. In the present multigrid algorithm the same operators are used for both restrictions. To stay conservative the restriction uses the volume weighted average. For a coarse grid volume V_c the restriction is provided by:

$$w_m(V_c) = I_{fm}^w(w_f)\Big|_{V_c} = \frac{1}{|V_c|} \sum_{i \in children(V_c)} |V_i| \cdot w_f(V_i)$$

where

$$|V_c| = \sum_{i \in children(V_c)} |V_i|.$$

The relationship of the coarse grid volumes and their children volumes is described in Part 1.7.2.

For the second restriction operator I^r the formula is the same, but some computations can be saved by summing up the fluxes instead of the residuals according to equation 2.51.

On boundaries of the coarse grids a violation of the respective boundary conditions has to be avoided. As described in Part 3.7, two approaches in the treatment of boundaries can be distinguished. Some boundary conditions can be fulfilled by computing the correct flux for the respective boundary faces. For these boundaries the residuals can be used as derived from the discretization on both the fine and the coarse grid.

For other boundaries the flow quantities and the fluxes have to be modified. The modification of the flow quantities has to be performed directly after the transfer to the coarser grid. For these volumes, certain fluxes are computed in the discretization. In correspondence to Part 3.7 the evaluated fluxes also have to be adapted to the respective boundary conditions before the forcing term can be computed.

2.5.13 Prolongation of the Corrections

The changes in the flow conditions obtained in the execution of a time step and by the consideration of corrections coming from the coarser grids have to be transferred as corrections to the finer grids using proper interpolation operators. The most trivial approach is the direct injection. The coarse grid correction of a volume V_c is directly transfered to each child volume V_f of V_c:

$$I^c_{mf}(C_m)\big|_{V_f} = C_m|_{V_c}$$

Further improvements may be achieved by employing higher order interpolation operators[1]. One possibility is the linear reconstruction of the corrections over the coarse grid control volumes. Therefore the gradients of the corrections are calculated on the coarse grid analog to Part 2.3.3 and limited according to Part 2.3.5. The points of evaluation and limitation for these reconstructed values on each coarse grid volume are the points attached to each of its children volume.

2.5.14 Smoothing Operators

The smoothing operators for the residuals are described in the Part 2.5.4. The smoothing of the corrections operates the same way after transferring the corrections to the finer grid. The changes in the flow quantities which are caused by the consideration of the corrections may lead to a violation of the boundary conditions. The initialization of a velocity on a viscous wall e.g. is not allowed. Therefore, the corrections on the boundary volumes have to be adopted to the respective boundary conditions. The method equals the approach as described for the treatment of the restrictions on the boundary volumes, where in this case instead of the fluxes the corrections have to be modified.

Finally, the possibility to smooth the flow quantities on a grid after the prolongation by employing one or more Runge-Kutta time steps should be mentioned as well as the smoothing effect of using only first order terms on the coarser grids.

2.5.15 Calculation of the Time Step Size

For the control volume surrounding point $P(j1)$ in Figure 2.3 time step is given by

$$\Delta t(j1) = \min(\ \Delta t(j1)^c\ ;\ \Delta t(j1)^V\ ) \tag{2.68}$$

[1]Together with some smoothing the convergence results on curved stretched grids are better with the simple direct injection than by using the linear reconstruction.

where $\Delta t(j1)^c$ is the convective time step and $\Delta t(j1)^V$ is the viscous time step. The convective time step $\Delta t(j1)$ with $\{\Delta t(m) \,|\, m = 1, \ldots N^P\}$ has to be determined as:

$$\Delta t(j1) = \mathrm{CFL} \max \cdot \frac{1}{\lambda^c(j1)} \tag{2.69}$$

where $\lambda^c(j1)$ with $\{\lambda^c(m) \,|\, m = 1, \ldots N^P\}$ denotes the maximum eigenvalue of the flux Jacobian and *CFL* the Courant number. The eigenvalue can be determined in an integration over the surface of the control volume around point $P(j1)$ with $n-1$ neighboring points $P(j2) \ldots P(jn)$. For a control volume as it is shown in Figure 2.10, the convective eigenvalues is:

$$\lambda^c(j) = \sum_{i=1}^{n-1} |\vec{v}(i) \cdot \vec{F}(i)| + a(i) \cdot |\vec{F}(i)| \tag{2.70}$$

with $\vec{F}(i)$ representing the face vectors of the control volume face for the kth neighbor of $P(j1)$ and $\vec{v}(i)$ the face velocity vector. Let face $F(i)$ be the interface between two control volumes around $P(j1) = C_F^P(i,1)$ and $P(j2) = C_F^P(i,2)$, see Figure 2.10. The face values $\vec{v}(i)$ and $a(i)$ are computed by an arithmetic averaging of the respective point values:

$$\vec{v}(i) = \frac{1}{2} \cdot \left[\begin{pmatrix} u(j1) \\ v(j1) \\ w(j1) \end{pmatrix} + \begin{pmatrix} u(j2) \\ v(j2) \\ w(j2) \end{pmatrix} \right]$$

and

$$a(i) = \frac{1}{2} \cdot \left(\sqrt{\frac{p(j1)}{\rho(j1)}} + \sqrt{\frac{p(j2)}{\rho(j2)}} \right).$$

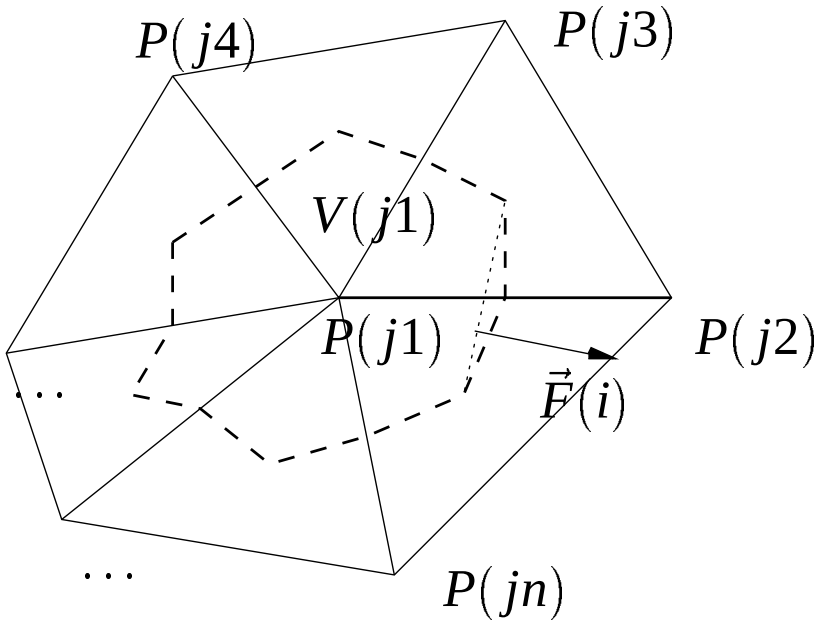


Figure 2.10: Face of a three dimensional control volume

The viscous time step $\Delta t(j1)^V$ to be scaled with a factor $K^v = 0.25$ (see [46]):

$$\Delta t^V(j1) = \mathrm{CFL} \cdot K^v \cdot \frac{1}{\lambda^V(j1)} \cdot V(j1).$$

Again, the eigenvalue can be determined in an integration over the surface of the control volume around point $P(j1)$ with $n-1$ neighboring points $P(j2)\dots P(jn)$. For a control volume as it is shown in Figure 2.10, the viscous eigenvalue is:

$$\lambda^V(j) = \sum_{i=1}^{n-1} (\lambda^{V,1}(i) + \lambda^{V,2}(i)) \frac{|\vec{F}(i)|^2}{\rho} \tag{2.71}$$

with

$$\lambda^{V,1}(i) = \frac{4}{3}(\mu_{lam}(i) + \mu_t(i))$$
$$\lambda^{V,2}(i) = \left(1 + \frac{Pr}{Pr_t}\frac{\mu_t(i)}{\mu_{lam}(i)}\right)\frac{\kappa_{lam}(i)}{c_v}$$

where

$$\mu(i) = \frac{1}{2}(\mu(i1) + \mu(i2)) \ , \quad \mu_t = \frac{1}{2}(\mu_t(i1) + \mu_t(i2)) \ , \ \text{etc.}$$

2.5.16 Computation of the Global Residual

After each time step the $\mathcal{L}_2$ residual is computed in order to monitor the convergence of the solution process. For each point $P(j)$ the local residual $(\rho_t)(j)$ is calculated as:

$$(\rho_t)(j) = \frac{\rho(j) - \rho^{(0)}(j)}{\Delta t(j) \cdot V(j)}$$

with $\rho(j)$ being the current density and $\rho^{(0)}(j)$ the density before the time step in point $P(j)$. The square of the residual weighted by the volume of the point is added up to calculate the square of the entire residual:

$$(\rho_t)^2 = \sum_{i=1}^{N^P} (\rho_t)(j)^2 \cdot V(j).$$

Herein N^P denotes the number of points and $V(j)$ is the size of the control volume around point $P(j)$.

2.6 Solution techniques for time-accurate computations

The temporal variation of the flow quantities can be written in general form for a point $P(j1)$ as:

$$\frac{d}{dt}\vec{W}(j1) + \vec{R}(j1) = 0. \tag{2.72}$$

A comparison with equation 2.8 gives for the residual $\vec{R}(j1)$:

$$\vec{R}(j1) = \frac{1}{V(j1)} \cdot \vec{Q}^F(j1). \tag{2.73}$$

For time-accurate computations, the TAU-Code provides currently the following options

- global time stepping
- dual time stepping.

2.6.1 Global time stepping scheme

The global time stepping scheme for the time-dependent problem (2.72) is based on the explicit K-step low-storage Runge-Kutta scheme (2.74) (as described by Jameson [33]):

$$\begin{array}{lcl} \vec{W}^{(0)}(j1) & = & \vec{W}(j1)(n) \\ \vec{W}^{(1)}(j1) & = & \vec{W}(j1)^{(0)} - \alpha(j)\Delta t(j1)\vec{R}_v^{(0)}(j1) \\ & \vdots & \\ \vec{W}^{(a)}(j1) & = & \vec{W}(j1)^{(0)} - \alpha_a\Delta t(j1)\vec{R}_v^{(a-1)}(j1) \\ \vec{W}(j1)(n+1) & = & \vec{W}^{(a)}(j1) \end{array} \tag{2.74}$$

with

$$\vec{R}_v(j1) = V(j1)\cdot\vec{R}(j1).$$

In this equation the residual $\vec{R}(j1)$ equals the fluxes $\vec{Q}(j1)^F$ over the control volume boundaries. For multigrid calculations the residual $\vec{R}(j1)$ is the sum of the fluxes $\vec{Q}^F(j1)$ over the boundaries of the control volume and the forcing term $\vec{Q}^P(j1)$ coming from the next finer grid. Details concerning the multigrid algorithm can be found in Part 2.5.7.

Therein, the time step width Δt is the minimum time step over all dual cells. To be more precise, let $\Delta t(j)$ denote the local time step for dual cell j, see Part 2.5.15 for its calculation. Then in global time stepping we set $\Delta t = \min_{j=1}^{N_c}\{\ \Delta t(j)\ \}$, where N_c denotes the number of dual grid cells.
In case of global time stepping, the number Runge-Kutta steps K has to be 1 or 2. Higher numbers violate the time accuracy. Moreover, the multigrid cycle must be "sg" for global timestepping, because multigrid in TAU is not time accurate.

2.6.2 Dual-time stepping scheme

Denote $[0,T]$ the time-interval and $t_0 = 0 < t_1 < \ldots < t_N = T$ a partition of $[0,T]$. We consider the following time-dependent problem

$$\frac{d\vec{W}(j1)}{dt} = -\vec{\mathcal{R}}(\vec{W})(j1) \tag{2.75}$$

where the notation $\vec{\mathcal{R}}(\vec{W})(j1)$ indicates that the residual $\vec{\mathcal{R}}(j1)$ for dual cell $j1$ was computed using the vector of conservative variables $\vec{W}$.
In a first step, a Backward Difference Formula (BDF) for discretizing the time-derivative is employed to (2.75). The TAU-Code provides BDF of first, second and third order of accuracy. For example, the second order accurate BDF reads

$$\frac{3}{2\Delta t}\vec{W}(j1)^{n+1} - \frac{4}{2\Delta t}\vec{W}(j1)^{n} + \frac{1}{2\Delta t}\vec{W}(j1)^{n-1} = -\vec{\mathcal{R}}(\vec{W}^{n+1})(j1) \tag{2.76}$$

where $\vec{W}(j1)^{\nu}$ denotes the solution at time t_ν.
We arrive at a sequence of (nonlinear) steady-state problems. One approach for the iterative solution of the nonlinear steady-state problem (2.76) is the so-called Dual-Time stepping scheme. We assume that $\vec{W}(j1)^n$, $\vec{W}(j1)^{n-1}$ have already been computed and we seek $\vec{W}(j1)^{n+1}$. For this purpose we consider the following equation in fictitious pseudo time t^*, viz.,

$$\frac{d\vec{W}(j1)^{n+1}}{dt^*} = -\vec{\mathcal{R}}^{DTS}(\vec{W}^{n+1})(j1) \tag{2.77}$$

with the modified residual

$$\vec{\mathcal{R}}^{DTS}(\vec{W}^{\nu})(j1) \equiv \vec{\mathcal{R}}(\vec{W}^{\nu})(j1) + \frac{3}{2\Delta t}\vec{W}(j1)^{\nu} - \frac{4}{2\Delta t}\vec{W}(j1)^{n} + \frac{1}{2\Delta t}\vec{W}(j1)^{n-1}$$

This problem can be integrated using a K-stage Runge-Kutta scheme until a steady state in fictitious pseudo time has been reached. Obviously the steady state solution of (2.77) is the solution of (2.76).
The integration in fictitious pseudo time using a k-stage Runge-Kutta scheme can be accelerated using the acceleration techniques for steady-state problems, see Part 2.5.

2.7 Initialization of the Flow Conditions

Before the solution process starts, the initial flow conditions are to be determined. In case of a restart (i.e. when a restart filename is defined in the input) they are read from a file. In case of starting a new computation initial values are assigned to all points of the grid. In the case that a farfield boundary exists in the computational domain the farfield values are considered as initial values, otherwise the reference values are used, which are needed for the adimensionalization of the flow quantities (note: the default values at the farfield boundary are the reference values themselves, but user-specified farfield values can be defined in the list of parameters. Thus, the dimensionless initial conditions, in the case of using farfield values, are given by:

$$\begin{aligned} \rho_{init} &= \rho_\infty \\ p_{init} &= \rho_\infty \cdot p_\infty \\ u_{init} &= \cos(\alpha) \cdot \cos(\beta) \cdot M_\infty \cdot \sqrt{\gamma \cdot \frac{p_\infty}{\rho_\infty}} \\ v_{init} &= \sin(\beta) \cdot M_\infty \cdot \sqrt{\gamma \cdot \frac{p_\infty}{\rho_\infty}} \\ w_{init} &= \sin(\alpha) \cdot \cos(\beta) \cdot M_\infty \cdot \sqrt{\gamma \cdot \frac{p_\infty}{\rho_\infty}} \end{aligned} \tag{2.78}$$

depending on the input for the onflow angle α and β. In case of viscous/turbulent computations the laminar/turbulent viscosity is:

$$\begin{aligned} \mu_{l,init} &= \frac{\rho_\infty \cdot u_\infty \cdot x_{\mathrm{Re}}}{\mathrm{Re}} \\ \mu_{t,init} &= \mu_{l,init} \cdot \mu_{t,ratio}. \end{aligned} \tag{2.79}$$

The factor $\mu_{t,ratio}$ is an input parameter (with default value). In the case of 2-equation turbulence models the initial turbulent kinetic energy k is computed from u, v, w_{init} with a given turbulent intensity (default or user input). The second transported quantity (which depends on the turbulence model) is then computed by taking the model dependent relation of it to k and $\mu_{t,init}$. A computation with e.g. a 2-equation turbulence model can be started using a restart file from a solution obtained with a another model, even with a 1-equation turbulence model (and vice versa). In each of the different possible cases the available quantities in the restart file are used for the computation of an initial guess for the unknown quantities. E.g. an initial guess for k is calculated with the velocity gradients and the Bradshaw assumption and some empirical factors, the second quantity then is approximated using the obtained distribution for k and the μ_t.

2.7.1 Reference Quantities

The reference quantities are used for the adimensionalization and as default values for a farfield boundary. Reference quantities having a default value are: (dimensional values are denoted by a bar):

Parameter:		Default Value:
Temperature	$\bar{T}_{ref}$	$273.15K$
Pressure	$\bar{p}_{ref}$	$101325\frac{N}{m^2}$
Mach number	M_{ref}	none
Reference velocity	$\bar{u}_{ref}$	none
Density	$\bar{\rho}_{ref}$	$\frac{\bar{p}_{ref}}{T_{ref}\cdot\Re}kg/m^3$
Reynolds Number	Re_{ref}	none
Reynolds Length	$\bar{x}_{\Re}$	$1m$
Ratio of Specific heats	γ	1.4
Prandtl Number	Pr_l	0.72
turb. Prandtl Number	Pr_t	0.9
Viscosity Ratio	μ_{ratio}	0.001

There is no default for the Reynold's number. Hence, for viscous calculations, the input of a value is required. No input for the Mach number is required if $\bar{u}_{ref}$ is defined:

$$M_{ref} = \frac{\bar{u}_{ref}}{\sqrt{\gamma\cdot\Re\cdot\bar{T}_{ref}}},$$

and vice versa. Note that the default dimensional values describe air at normal atmospheric conditions. If e.g. a wind tunnel experiment is calculated, the reference values have to be defined according to the experiment. Note that the Pressure has to be defined in N/m^2 and not in bar. Note that the definition of a reference value in the input parameter list overwrites the default. Note that an overdetermination of the reference quantities is not allowed. It is checked internally that the ideal gas assumption holds e.g. if M_{ref} and $\bar{u}_{ref}$ are defined in the input the values have to fulfill the formula: $M_{ref} = \frac{\bar{u}_{ref}}{\sqrt{\gamma\cdot\Re\cdot\bar{T}_{ref}}}$. The reference viscosities are computed from the given values:

$$\begin{aligned}\bar{\mu}_{l,ref} &= \frac{\bar{\rho}_{ref}\cdot\bar{u}_{ref}\cdot\bar{x}_{\mathrm{Re}}}{\mathrm{Re}},\\ \bar{\mu}_{t,ref} &= \bar{\mu}_{l,ref}\cdot\mu_{ratio}.\end{aligned} \tag{2.80}$$

2.7.2 Dimensionless Quantities

Internally, the code uses dimensionless quantities. The formulation that is realized in the code mainly follows the approach presented in [56]. The dimensionless values are determined using the reference quantities (2.7.1):

$$\begin{aligned}x &= \frac{\bar{x}}{\bar{x}_{\mathrm{Re}}},\\ y &= \frac{\bar{y}}{\bar{x}_{\mathrm{Re}}},\\ z &= \frac{\bar{z}}{\bar{x}_{\mathrm{Re}}},\\ \rho &= \frac{\bar{\rho}}{\bar{\rho}_{ref}},\\ u &= \bar{u}\cdot\sqrt{\frac{\bar{\rho}_{ref}}{\bar{p}_{ref}}},\\ v &= \bar{v}\cdot\sqrt{\frac{\bar{\rho}_{ref}}{\bar{p}_{ref}}},\\ w &= \bar{w}\cdot\sqrt{\frac{\bar{\rho}_{ref}}{\bar{p}_{ref}}},\\ p &= \frac{\bar{p}}{\bar{p}_{ref}},\\ T &= \frac{\bar{T}}{\bar{T}_{ref}}.\end{aligned} \tag{2.81}$$

Accordingly additional dimensionless values are computed as e.g. a dimensionless wall temperature T_W is obtained by:

$$T_W = \frac{\bar{T}_W}{\bar{T}_{ref}}. \tag{2.82}$$

The dimensionless reference viscosity and thermal conductivity can be determined as:

$$\begin{aligned} \mu_{l,ref} &= \frac{\sqrt{\gamma}\cdot M_{ref}}{\mathrm{Re}}, \\ \mu_{t,ref} &= \mu_{l,ref}\cdot\mu_{ratio} \\ \kappa_{l,ref} &= \frac{\mu_{l,ref}}{\mathrm{Pr}_l}\cdot\frac{\gamma}{\gamma-1}, \\ \kappa_{t,ref} &= \frac{\mu_{t,ref}}{\mathrm{Pr}_t}\cdot\frac{\gamma}{\gamma-1}. \end{aligned} \tag{2.83}$$

The local laminar viscosity μ_l can be determined from the current temperature employing the law of Sutherland:

$$\mu_l = \mu_{l,ref}\cdot\left(\frac{\bar{T}}{\bar{T}_{ref}}\right)^{1.5}\cdot\frac{\bar{T}_{ref}+110.4K}{\bar{T}+110.4K} = \mu_{l,ref}\cdot T^{1.5}\cdot\frac{1+110.4K/\bar{T}_{ref}}{T+110.4K/\bar{T}_{ref}}. \tag{2.84}$$

The local turbulent viscosity μ_t is obtained by a suited turbulence model. Also the thermal conductivity k can be computed using the Sutherland law. Thus, the ratios of the local thermal conductivity and viscosity and the farfield values are equal:

$$\frac{\mu_l}{\mu_{l,ref}} = \frac{\kappa_l}{\kappa_{l,ref}},$$

which leads to

$$\kappa_l = \frac{\mu_l}{\mathrm{Pr}_l}\cdot\frac{\gamma}{\gamma-1}. \tag{2.85}$$

The same relationship holds for the turbulent thermal conductivity:

$$\kappa_t = \frac{\mu_t}{\mathrm{Pr}_t}\cdot\frac{\gamma}{\gamma-1}.$$

Note that as $\rho_{ref}=1$, $p_{ref}=1$ and $T_{ref}=1$ the dimensionless gas constant R equals 1 as well. Thus, it does not have to be considered for computations with dimensionless variables. All output of flow quantities is performed in dimensional values. Hence, in order to write out the monitor output or a restart file, all dimensionless values have to be transferred to their dimensional form:

$$\begin{aligned} \bar{x} &= x\cdot\bar{x}_{\mathrm{Re}}, \\ \bar{y} &= y\cdot\bar{x}_{\mathrm{Re}}, \\ \bar{z} &= z\cdot\bar{x}_{\mathrm{Re}}, \\ \bar{\rho} &= \rho\cdot\bar{\rho}_{ref} \\ \bar{u} &= u\cdot\sqrt{\frac{\bar{p}_{ref}}{\bar{\rho}_{ref}}}, \\ \bar{v} &= v\cdot\sqrt{\frac{\bar{p}_{ref}}{\bar{\rho}_{ref}}}, \\ \bar{w} &= w\cdot\sqrt{\frac{\bar{p}_{ref}}{\bar{\rho}_{ref}}}, \\ \bar{p} &= p\cdot\bar{p}_{ref}, \\ \bar{\mu}_t &= \mu_t\cdot\frac{\bar{\mu}_{t,ref}}{\mu_{t,ref}}. \end{aligned} \tag{2.86}$$

As

$$\bar{\mu}_{t,ref} = \bar{\mu}_{l,ref}\cdot\mu_{ratio} \text{ and } \mu_{t,ref} = \mu_{l,ref}\cdot\mu_{ratio},$$

$\bar{\mu}_t$ can also be computed as

$$\bar{\mu}_t = \mu_t\cdot\frac{\bar{\mu}_{l,ref}}{\mu_{l,ref}}, \tag{2.87}$$

and $\bar{\mu}_{t,ref}$ is not needed.

3 Boundary Treatment

The boundaries of the computational domain are divided into several *boundary parts*, see also section 1.3.3. There is a certain *boundary type* related to each boundary part. Implemented boundary types are:

- axisymmetry axis
- actuation
- actuator inflow
- actuator exhaust
- chimera
- dirichlet
- engine inflow
- engine exhaust
- euler wall
- exit-pressure outflow
- farfield
- laminar wall
- mirror plane
- periodic plane
- reservoir-pressure inflow
- sharp edge
- supersonic inflow
- supersonic outflow
- symmetry plane
- turbulent wall
- viscous wall

In general there is one algorithm for each of the boundary treatments. The algorithms are described in the following.

Furthermore, for some boundary parts, as e.g. ”euler wall” or ”laminar wall” it is desirable to integrate forces (pressure and/or viscous forces) in order to determine the lift and drag coefficients. The determination of the pressure forces is described in section 3.5.

3.1 Inviscid Wall Flux Computation

In this section the evaluation of the convective/inviscid fluxes over the boundary faces lying on an ”Euler wall” is described.

Let $F_b(b,i)$ be a secondary boundary grid face of the boundary part b of the type ”Euler Wall”. The point related to the face is given by $P(j) = C^P_{F_b}(b,i)$. The convective fluxes for the momentum equation over $F_b(b,i)$ are given by the product of the normal vector of the boundary face $\vec{F}_b$ and the pressure on the wall. The fluxes are:

$$\vec{F}_i^c = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho Hu \end{pmatrix}, \vec{G}_i^c = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho Hv \end{pmatrix}, \vec{H}_i^c = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho Hw \end{pmatrix}. \tag{3.1}$$

The complete flux will become for inviscid fluxes, see (2.2):

$$\bar{\bar{F}} = \vec{F}_i^c \cdot \vec{n}_x + \vec{G}_i^c \cdot \vec{n}_y + \vec{H}_i^c \cdot \vec{n}_z \tag{3.2}$$

The slip-wall condition on every inviscid wall is defined as:

$$u n_x + v n_y + w n_z = 0. \tag{3.3}$$

Summing up the whole flux from (3.2) will lead to:

$$\bar{\bar{F}} = \begin{pmatrix} \rho\,(u n_x + v n_y + w n_z) \\ \rho u\,(u n_x + v n_y + w n_z) + p n_x \\ \rho v\,(u n_x + v n_y + w n_z) + p n_y \\ \rho w\,(u n_x + v n_y + w n_z) + p n_z \\ \rho H\,(u n_x + v n_y + w n_z) \end{pmatrix} = \begin{pmatrix} 0 \\ p n_x \\ p n_y \\ p n_z \\ 0 \end{pmatrix}. \tag{3.4}$$

Since the point $P(j)$ is lying on the wall the pressure can be taken directly from the point and finally the euler wall boundary condition becomes with the boundary grid face $F_b(b,i)$:

$$\vec{Q}^c_{F_b} = \begin{pmatrix} 0 \\ F_b^x(b,i) \cdot p(j) \\ F_b^y(b,i) \cdot p(j) \\ F_b^z(b,i) \cdot p(j) \\ 0 \end{pmatrix}. \tag{3.5}$$

To finish the flux computation for point $P(j)$ the boundary fluxes are added to the current fluxes computed for point $P(j)$.

3.2 Projection of Velocities

In this section the adaption of the flow quantities for the boundary points lying on a "symmetry plane" is described.

Let $F_b(b,i)$ be a boundary face of a boundary part b of the type "Symmetry Plane". The velocities in the boundary point $P(j) = C^P_{F_b}(b,i)$ have to be projected onto the symmetry plane. The direction of the plane is given by the normal vector $\vec{F}_b(b,i) = (F^x_b(b,i), F^y_b(b,i), F^z_b(b,i))^T$. The normal vector has to be normalized with

$$\vec{F}^n_b(i) = \begin{pmatrix} n^x_b(b,i) \\ n^y_b(b,i) \\ n^z_b(b,i) \end{pmatrix} = \frac{1}{|\vec{F}_b(b,i)|} \cdot \begin{pmatrix} F^x_b(b,i) \\ F^y_b(b,i) \\ F^z_b(b,i) \end{pmatrix}. \tag{3.6}$$

The corrected velocity vector $\vec{v}_{corr}(j) = ((u_{corr}(j), v_{corr}(j), w_{corr}(j))^T$ is calculated from the given velocity vector $\vec{v}_g(j) = (u_g(j), v_g(j), w_g(j))^T$ and the normalized normal vector $\vec{F}^n_b(i)$ by

$$\vec{v}_{corr} = \vec{v}_g - \vec{F}^n_b(i) \cdot \left(\vec{F}^n_b(i) \cdot \vec{v}_g \right) \tag{3.7}$$

The density and the inner energy of the flow must not be affected by the projection. Hence, the pressure in the point has to be corrected after the projection. Before the projection, the local product of density and energy, $\rho E(j)$ is computed as:

$$\rho E(j) = \frac{p_g(j)}{\gamma - 1} + \rho(j) \cdot \frac{1}{2} \cdot \left(u_g(j)^2 + v_g(j)^2 + w_g(j)^2 \right). \tag{3.8}$$

Then the velocities are projected and with the new velocities and the calculated product $\rho E(j)$, the pressure is corrected as:

$$p_{corr}(j) = (\gamma - 1) \cdot \left[\rho(j) \cdot E(j) - \rho(j) \cdot \frac{1}{2} \cdot \left(u_{corr}(j)^2 + v_{corr}(j)^2 + w_{corr}(j)^2 \right) \right]. \tag{3.9}$$

3.3 Farfield Boundary Flux Computation

At the farfield boundaries of the computational domain convective fluxes have to be determined crossing the boundary faces. The calculation of the fluxes is done using the AUSM Riemann solver. In order to calculate the correct fluxes, the flow conditions outside the boundary face have to be determined. They are calculated employing the theory of Whitfield [79]. Figure 3.1 depicts the situation at the farfield boundary. The reference point P_L is located on the boundary face, representing the flow quantities inside the computational domain, while the reference point P_R is located outside the domain.

Let $F_b(b,i)$ be a boundary face of a boundary part b of the type "Farfield Boundary". The fluxes over $F_b(b,i)$ are adjoined to the boundary point $P(j) = C^P_{F_b}(b,i)$ related to $F_b(b,i)$.

Determination of Flow Conditions at the Boundary Face

The orientation of the boundary face is given by the normal vector

$$\vec{F}_b(b,i) = (F^x_b(b,i), F^y_b(b,i), F^z_b(b,i))^T.$$

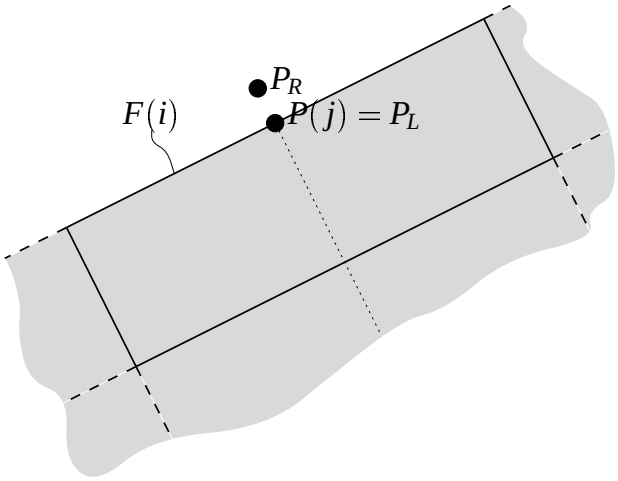


Figure 3.1: Situation at farfield boundary

The normal vector has to be normalized with

$$\vec{F}_b^n(i) = \begin{pmatrix} n_b^x(b,i) \\ n_b^y(b,i) \\ n_b^z(b,i) \end{pmatrix} = \frac{1}{|\vec{F}_b(b,i)|} \cdot \begin{pmatrix} F_b^x(b,i) \\ F_b^y(b,i) \\ F_b^z(b,i) \end{pmatrix}. \tag{3.10}$$

As the reference point P_L in the flow field is identical to the current boundary point $P(i)$, one can write $\vec{W}_L = \vec{W}(i)$. In order to determine the flow quantities at the outer side of the boundary face (index R), the following values have to be derived from the primitive variables in the reference point P_L:

$$vn_L = \vec{v}_L \cdot \vec{F}_b^n(i) \tag{3.11}$$

$$E_L = \frac{p_L}{\rho_L \cdot (\gamma - 1)} \tag{3.12}$$

$$H_L = E_L + \frac{1}{2} \cdot \left(u_L^2 + v_L^2 + w_L^2\right) + \frac{p_L}{\rho_L} \tag{3.13}$$

$$a_L = \sqrt{(\gamma - 1) \cdot H_L - \frac{1}{2} \cdot \left(u_L^2 + v_L^2 + w_L^2\right)} \tag{3.14}$$

$$M_L = \frac{vn_L}{a_L} \tag{3.15}$$

with $\vec{v}_L = (u_L, v_L, w_L)^T$. The linearization point P_0 is assumed to lie on the boundary, therefore one can write

$$\rho_0 = \rho_L \quad \text{and} \quad a_0 = a_L.$$

3.3.1 Supersonic Inflow/Outflow

If the absolute value of M_L is greater than 1, the face is lying on a supersonic inflow or supersonic outflow boundary. Depending on the sign of M_L, the flow conditions are either equal to the conditions in the reference point p_L (outflow, $M_L > 0$) or to the farfield values (inflow, $M_L < 0$):

$$\vec{W}_R = \begin{cases} \vec{W}_\infty & \text{for} \quad M_L \leq -1 \\ \vec{W}_L & \text{for} \quad M_L > 1 \end{cases}$$

If the absolute value of M_L is smaller than one, the face is lying on a subsonic boundary, for that one can also distinguish between inflow and outflow, depending on the sign of M_L.

3.3.2 Subsonic Outflow

For the subsonic outflow case, the conditions for the flow approaching the boundary (index app) is given by the flow conditions in the reference point P_L:

$$\vec{W}_{app} = \vec{W}_L$$

while the conditions for the flow leaving the boundary (index lve) are given by the infinity values:

$$\vec{W}_{lve} = \vec{W}_\infty$$

The pressure on the boundary $p(j)$ equals the pressure in the leaving flow:

$$p_{bnd} = p_{lve} = p_L. \tag{3.16}$$

After having determined the leaving and approaching flow conditions and the boundary pressure, the flow conditions on the outer side of the boundary can be determined as

$$\begin{aligned} \rho_R &= \rho_{app} + \frac{p_{bnd} - p_{app}}{a_0^2} \\ u_R &= u_{app} - n_b^x(b,i) \cdot \frac{p_{app} - p_{bnd}}{\rho_0 \cdot a_0} \\ v_R &= v_{app} - n_b^y(b,i) \cdot \frac{p_{app} - p_{bnd}}{\rho_0 \cdot a_0} \\ w_R &= w_{app} - n_b^z(b,i) \cdot \frac{p_{app} - p_{bnd}}{\rho_0 \cdot a_0} \\ p_R &= p_{bnd} \end{aligned} \tag{3.17}$$

3.3.3 Subsonic Inflow

In the subsonic inflow case the values for the flow leaving the boundary are the flow conditions in the reference point:

$$\vec{W}_{lve} = \vec{W}_L$$

while for the conditions of the flow approaching the boundary the infinity values are taken:

$$\vec{W}_{app} = \vec{W}_\infty$$

The boundary pressure p_{bnd} is given by the relation:

$$p_{bnd} = \frac{1}{2} \cdot \left[p_{app} + p_{lve} + \rho_0 \cdot a_0 \cdot \begin{pmatrix} n_b^x(b,i) \\ n_b^y(b,i) \\ n_b^z(b,i) \end{pmatrix} \cdot \left[\begin{pmatrix} u \\ v \\ w \end{pmatrix}_{lve} - \begin{pmatrix} u \\ v \\ w \end{pmatrix}_{app} \right] \right] \tag{3.18}$$

After having determined the leaving and approaching flow conditions and the boundary pressure, the flow conditions on the outer side of the boundary can be determined as

$$\begin{aligned} \rho_R &= \rho_{app} + \frac{p_{bnd} - p_{app}}{a_0^2} \\ u_R &= u_{app} + n_b^x(b,i) \cdot \frac{p_{app} - p_{bnd}}{\rho_0 \cdot a_0} \\ v_R &= v_{app} + n_b^y(b,i) \cdot \frac{p_{app} - p_{bnd}}{\rho_0 \cdot a_0} \\ w_R &= w_{app} + n_b^z(b,i) \cdot \frac{p_{app} - p_{bnd}}{\rho_0 \cdot a_0} \\ p_R &= p_{bnd} \end{aligned} \tag{3.19}$$

The flux over the boundary is computed using the AUSM Scheme as described in Part 2.3.1.

3.3.4 Nacelle Inflow Flux Computation

The simulation of the nacelle inflow is performed according to Rudnik [60]. Input parameter is the mass flow $\dot{m}_{ref}$. With a known fan area A_{Fan} (geometric data, see Part 1.5.2), the area relation $\varepsilon_{Fan} = A_\infty / A_{Fan}$ can be computed as:

$$\varepsilon_{Fan} = \frac{\dot{m}_{ref}}{A_{Fan} \cdot M_\infty \cdot \sqrt{\gamma \cdot p_\infty \cdot \rho_\infty}} \tag{3.20}$$

From the free stream Mach number the free stream Laval number La_∞ can be computed:

$$La_\infty = \sqrt{\frac{(\gamma+1) \cdot M_\infty^2}{(\gamma-1) \cdot M_\infty^2 + 2}}. \tag{3.21}$$

The free stream Mach number has to be determined from the free stream velocity u_∞ and the free stream pressure and density:

$$M_\infty = u_\infty \cdot \sqrt{\frac{\rho_\infty}{\gamma \cdot p_\infty}}.$$

The critical flow area relation A^*/A_∞ can be determined from the equation

$$\frac{A^*}{A_\infty} = La_\infty \cdot \left[\frac{1 - \frac{\gamma-1}{\gamma+1} \cdot La_\infty^2}{1 - \frac{\gamma-1}{\gamma+1}} \right]^{\frac{1}{\gamma-1}}. \tag{3.22}$$

Employing the product of the critical area relation and ε_{Fan} the fan Laval number La_{Fan} is computed from the transcendent equation

$$\varepsilon_{Fan} \cdot \frac{A^*}{A_\infty} = La_{Fan} \cdot \left[\frac{1 - \frac{\gamma-1}{\gamma+1} \cdot La_{Fan}^2}{1 - \frac{\gamma-1}{\gamma+1}} \right]^{\frac{1}{\gamma-1}}. \tag{3.23}$$

This equation is solved iteratively. The value of La_{Fan} of the last time step (initialized with zero for the first time step) is used as a first estimate. Employing equation 3.23 the corresponding product $(\varepsilon_{Fan} \cdot \frac{A^*}{A_\infty})_{est}$ can be calculated.

The difference ΔA with

$$\Delta A = (\varepsilon_{Fan} \cdot \frac{A^*}{A_\infty})_{corr} - (\varepsilon_{Fan} \cdot \frac{A^*}{A_\infty})_{est}$$

multiplied with the derivative $(\varepsilon_{Fan} \cdot \frac{A^*}{A_\infty})'_{est}$ and scaled by a relaxation factor 0.5 gives the value ΔLa_{Fan} the current La_{Fan} has to be changed:

$$La_{Fan}(n+1) = \frac{1}{2} \cdot (La_{Fan}(n) - \Delta La_{Fan})$$

The derivative can be calculated as:

$$(\varepsilon_{Fan} \cdot \frac{A^*}{A_\infty})'_{est} = \frac{1}{(1 - \frac{\gamma-1}{\gamma+1})^{\frac{1}{\gamma-1}}} \cdot (1 - \frac{\gamma-1}{\gamma+1} \cdot La_{Fan}^2)^{\frac{2-\gamma}{\gamma-1}} \cdot (1 - \frac{\gamma-1}{\gamma+1} \cdot La_{Fan}^2 \cdot (1 + 2 \cdot \frac{1}{\gamma-1}))$$

The iterative process is executed until the difference ΔA becomes smaller than a certain predefined value ε. When the difference has become smaller the current La_{Fan} is taken as the result from equation 3.23.

With the fan Laval number the Mach number at the fan inflow plane M_{Fan} can be determined as:

$$M_{Fan} = \sqrt{\frac{2 \cdot La_{Fan}^2}{(\gamma+1) - (\gamma-1) \cdot La_{Fan}^2}} \tag{3.24}$$

The speed of sound at the fan face is given by

$$a_{Fan} = \frac{a_{0,\infty}}{\sqrt{1 + \frac{\gamma-1}{2} \cdot M_{Fan}^2}} \tag{3.25}$$

while the corresponding pressure can be computed as

$$p_{Fan} = \frac{p_{0,\infty}}{\left(1 + \frac{\gamma-1}{2} \cdot M_{Fan}^2\right)^{\frac{\gamma}{\gamma-1}}}. \tag{3.26}$$

The stagnation free stream values of the speed of sound $a_{0,\infty}$ and the pressure $p_{0,\infty}$ are derived from:

$$\begin{aligned} a_{0,\infty} &= a_\infty \cdot \sqrt{1 + \frac{\gamma-1}{2} \cdot M_\infty^2} \\ p_{0,\infty} &= p_\infty \cdot \left(1 + \frac{\gamma-1}{2} \cdot M_\infty^2\right)^{\frac{\gamma}{\gamma-1}} \\ u_{Fan} &= M_{Fan} \cdot a_{Fan}. \end{aligned}$$

From these values the remaining flow quantities at the fan face can be computed:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix}_{Fan} = u_{Fan} \cdot \vec{S}_{Fan} \tag{3.27}$$

$$\rho_{Fan} = \rho_\infty \cdot \left(\frac{p_{Fan}}{p_\infty}\right)^{\frac{1}{\gamma}} \tag{3.28}$$

The vector $\vec{S}_{Fan}$ describes a unit vector normal to the inflow plane. The determined quantities are assigned to all points lying on the fan face boundary part.

3.4 Nacelle Exhaust Flux Computation

Input data for the calculation of the flow quantities at the nacelle outflow faces are the pressure ratio $p_{0,Ex}/p_\infty$ and the temperature ratio $T_{0,Ex}/T_{0,\infty}$.

From the given ratios the stagnation exhaust temperature $T_{0,Ex}$ and the stagnation pressure $p_{0,Ex}$ can be determined:

$$\begin{aligned} T_{0,Ex} &= \left(\frac{T_{0,Ex}}{T_{0,\infty}}\right) \cdot T_{0,\infty} \\ p_{0,Ex} &= \left(\frac{p_{0,Ex}}{p_\infty}\right) \cdot p_\infty. \end{aligned}$$

The stagnation farfield temperature is given by:

$$T_{0,\infty} = \frac{p_\infty}{\rho_\infty \cdot \Re} \cdot \left(\frac{p_{0,\infty}}{p_\infty} \right)^{\frac{\gamma-1}{\gamma}},$$

while the stagnation density at the exhaust plane can be computed as:

$$\rho_{0,Ex} = \frac{p_{0,Ex}}{\Re \cdot T_{0,Ex}}.$$

Let $F_b(b,i)$ be a boundary face of a boundary part b of the type "Nacelle Exhaust". The flow variables in the boundary point $P(j1) = C^P_{F_b}(b,i)$ related to $F_b(b,i)$ are computed employing the reference point $P(j2) = C^{P_r}_{F_b}(b,i)$.

Employing some isentropic relationships the flow quantities can now be determined:

$$\begin{aligned} \rho(j1) &= \rho_{0,Ex} \cdot \left(\frac{p(j2)}{p_{0,Ex}} \right)^{\frac{1}{\gamma}} \\ \begin{pmatrix} u(j1) \\ v(j1) \\ w(j1) \end{pmatrix} &= \sqrt{\frac{2\cdot\gamma}{\gamma-1} \cdot \left(\frac{p_{0,Ex}}{\rho_{0,Ex}} - \frac{p(j2)}{\rho(j1)} \right)} \cdot \vec{F}_b(b,i) \cdot \frac{1}{|\vec{F}_b(b,i)|} \end{aligned}$$

The contribution to the mass flow over the exhaust plane can be computed as:

$$(\dot{m}_{ref})_{F_b} = \vec{F}_b(b,i) \cdot \begin{pmatrix} u(j1) \\ v(j1) \\ w(j1) \end{pmatrix} \cdot \rho(j1)$$

The resulting mass flow is composed of contributions coming from all exhaust boundary faces (lying on the fan and core plane). The mass flow is then used as an input value for the determination of the flow quantities at the fan inflow face (see section 3.3.4). Therefore it is essential that the quantities at the exhaust planes are determined before the inflow values can be computed.

3.5 Calculation of Pressure Forces

In order to compute the global lift and drag coefficients the forces implied by the pressure on the contour are determined for the boundary parts on the surface of the configuration.

Let $F_b(b,i)$ be a secondary boundary grid face of the boundary part b located on the surface of the configuration. The point related to the face is given by $P(j) = C^P_{F_b}(b,i)$. Figure 3.2 depicts the situation at the boundary. The size and the orientation of the boundary face is given by the face normal vector $\vec{F}_b(b,i)$. As can be seen in Figure 3.2 the vector of pressure forces $\vec{S}_p(b,i) = (S^x_p(b,i), S^z_p(b,i), S^y_p(b,i))^T$ with $\{\vec{S}_p(m,n) \,|\, m = 1,\ldots N^b, n = 1,\ldots N^{F_b}(b)\}$ for face $F_b(b,i)$ equals the product of normal vector $\vec{F}_b(b,i)$ and the pressure $p(j)$ in the respective boundary point:

$$\vec{S}_p(b,i) = \begin{pmatrix} F^x_b(b,i) \\ F^y_b(b,i) \\ F^z_b(b,i) \end{pmatrix} \cdot p(j) \tag{3.29}$$

The forces are summed up in the three coordinate directions for the entire boundary part. All positive components of the normal vectors in z-directions are also summed up in order to calculate the projection $A_z(b)$ with $\{A_z(m) \,|\, m = 1,\ldots N^b\}$ of the surface to a plane with $z = const$:

$$A_z(b) = \sum_{i=1}^{N^b} \left[F^z_b(b,i) \;\text{ if }\; F^z_b(b,i) > 0 \right]$$

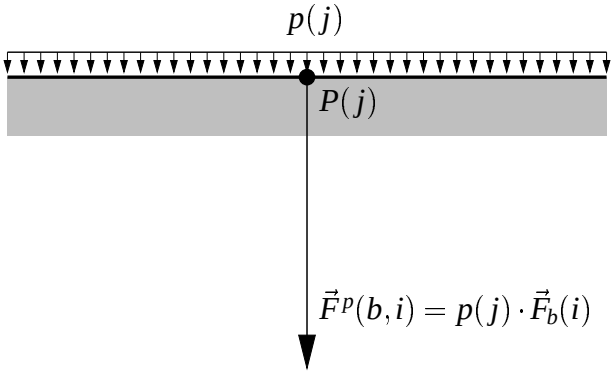


Figure 3.2: Computation of pressure forces

The drag coefficient $c_d(b)$ with $\{c_d(m) \,|\, m = 1, \ldots N^b\}$ for the current boundary part b can then be calculated as

$$c_d(b) = \frac{4 \cdot \left(\cos\alpha \cdot S_p^x(b,i) + \sin\alpha \cdot S_p^z(b,i)\right)}{u_{ref}^2 \cdot \rho_{ref} \cdot A_z(b)}, \tag{3.30}$$

while the lift coefficient c_l with $\{c_l(m) \,|\, m = 1, \ldots N^b\}$ can be calculated as

$$c_l(b) = \frac{4 \cdot \left(-\sin\alpha \cdot S_p^x(b,i) + \cos\alpha \cdot S_p^z(b,i)\right)}{u_{ref}^2 \cdot \rho_{ref} \cdot A_z(b)}. \tag{3.31}$$

In these equations α denotes the incidence, while u_{ref} and ρ_{ref} denote the infinity values of velocity and density.

Since in general there is more than one boundary part that contributes to the pressure forces, the lift and drag coefficient of all respective boundary parts are summed up to form the global coefficients.

3.6 Calculation of Viscous Forces

In order to compute the viscous contributions for the global lift and drag coefficient the forces implied by the friction on the boundary are to be taken into account. The principle is the same as it is for the calculation of the pressure forces (see 3.5), except the formulation of the vector of forces differ. This vector is defined now

$$\vec{S}_v(b,i) = \begin{pmatrix} \tau_{xx}\tau_{xy}\tau_{xz} \\ \tau_{yx}\tau_{yy}\tau_{yz} \\ \tau_{zx}\tau_{zy}\tau_{zz} \end{pmatrix} \cdot \begin{pmatrix} F_b^x(b,i) \\ F_b^y(b,i) \\ F_b^z(b,i) \end{pmatrix} \tag{3.32}$$

with τ_{ij} being the tensor defining the viscous stresses.

3.7 Realization of Boundary Conditions

One can distinguish between two groups of boundary conditions: the types of the first group require only the computation of an additional flux over the boundary face. This flux is adjoined to the respective boundary points and treated as the interior fluxes. These boundary fluxes are

computed together with the interior fluxes. The boundary types of this group are the Euler wall and the farfield boundary.

The other group of boundary types include the symmetry plane and the engine inflow/exhaust boundaries. In order to fulfill the conditions at these boundaries, the flow quantities in the boundary points have to be modified. The modification of the quantities is performed before the computation of the interior and the boundary fluxes. Hence, the modified flow conditions can already be used in the computation of fluxes between the boundary points and their neighbors.

In order to avoid the change of the flow conditions that may violate the boundary conditions during the time integration step, the fluxes that are computed for these points have to be modified. The fluxes of the momentum equations for the symmetry plane points are projected to the plane, while the fluxes of all equations have to be set to zero for the points on engine inflow/exhaust boundaries. The modification of the fluxes has to be performed after the interior and boundary fluxes have been determined for the entire grid.

4 Turbulence models

4.1 The turbulent equations

The considered equations for turbulent flows are the mass-weighted (Favre) averaged Navier-Stokes equations. Mass weighted quantities are indicated here by an overline bar. This leads to the notation:

$$\frac{\partial}{\partial t}\iiint_V \vec{\bar{W}}\,dV = -\iint_{\partial V} \vec{\bar{F}}\cdot\vec{n}\,dS \tag{4.1}$$

with the vector $\vec{\bar{W}}$ representing the averaged conserved quantities

$$\vec{\bar{W}} = \begin{pmatrix} \bar{\rho} \\ \bar{\rho}\bar{u} \\ \bar{\rho}\bar{v} \\ \bar{\rho}\bar{w} \\ \bar{\rho}\bar{E} \end{pmatrix}, \tag{4.2}$$

and the inviscid (indices i) and viscous (indices v) fluxes

$$\vec{\bar{F}} = \vec{\bar{F}}_i + \vec{\bar{F}}_v,\ \vec{\bar{G}} = \vec{\bar{G}}_i + \vec{\bar{G}}_v,\ \vec{\bar{H}} = \vec{\bar{H}}_i + \vec{\bar{H}}_v. \tag{4.3}$$

The flux vectors are:

$$\vec{\bar{F}}_i = \begin{pmatrix} \bar{\rho}\bar{u} \\ \bar{\rho}\bar{u}^2 + \bar{p} \\ \bar{\rho}\overline{uv} \\ \bar{\rho}\overline{uw} \\ \bar{\rho}\bar{H}\bar{u} \end{pmatrix}, \quad \vec{\bar{F}}_v = -\begin{pmatrix} 0 \\ \bar{\tau}_{xx} - \overline{\rho(u')^2} \\ \bar{\tau}_{xy} - \overline{\rho u'v'} \\ \bar{\tau}_{xz} - \overline{\rho u'w'} \\ \bar{u}\bar{\tau}_{xx} + \bar{v}\bar{\tau}_{xy} + \bar{w}\bar{\tau}_{xz} + \kappa_l\frac{\partial \bar{T}}{\partial x} - \frac{\partial \overline{\rho H'u'}}{\partial x} \end{pmatrix}, \tag{4.4}$$

$$\vec{\bar{G}}_i = \begin{pmatrix} \bar{\rho}\bar{v} \\ \bar{\rho}\overline{uv} \\ \bar{\rho}\bar{v}^2 + \bar{p} \\ \bar{\rho}\overline{vw} \\ \bar{\rho}\bar{H}\bar{v} \end{pmatrix}, \quad \vec{\bar{G}}_v = -\begin{pmatrix} 0 \\ \bar{\tau}_{xy} - \overline{\rho u'v'} \\ \bar{\tau}_{yy} - \overline{\rho(v')^2} \\ \bar{\tau}_{yz} - \overline{\rho v'w'} \\ \bar{u}\bar{\tau}_{xy} + \bar{v}\bar{\tau}_{yy} + \bar{w}\bar{\tau}_{yz} + \kappa_l\frac{\partial \bar{T}}{\partial y} - \frac{\partial \overline{\rho H'v'}}{\partial y} \end{pmatrix}, \tag{4.5}$$

$$\vec{\bar{H}}_i = \begin{pmatrix} \bar{\rho}\bar{w} \\ \bar{\rho}\overline{uw} \\ \bar{\rho}\overline{vw} \\ \bar{\rho}\bar{w}^2 + \bar{p} \\ \bar{\rho}\bar{H}\bar{w} \end{pmatrix}, \quad \vec{\bar{H}}_v = -\begin{pmatrix} 0 \\ \bar{\tau}_{xz} - \overline{\rho u'w'} \\ \bar{\tau}_{yz} - \overline{\rho y'w'} \\ \bar{\tau}_{xx} - \overline{\rho(w')^2} \\ \bar{u}\bar{\tau}_{xz} + \bar{v}\bar{\tau}_{yz} + \bar{w}\bar{\tau}_{zz} + \kappa_l\frac{\partial \bar{T}}{\partial z} - \frac{\partial \overline{\rho H'w'}}{\partial z} \end{pmatrix}. \tag{4.6}$$

Following the eddy viscosity hypothesis the Reynolds stress tensor can be formulated as:

$$-\overline{\rho u_i' u_j'} = \mu_t \cdot \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial \bar{u}_k}{\partial x_k}\right) - \frac{2}{3}\delta_{ij}\bar{\rho}k \tag{4.7}$$

The last term in this equation contains the turbulent kinetic energy k, which is

$$k = \frac{\overline{u_i'}\overline{u_i'}}{2} \tag{4.8}$$

The turbulent fluxes $-\overline{\rho H' u_i'}$ can be expressed by the heat fluxes $-\overline{\rho u_i' T'}$ and an extra term composed of a product of velocity fluctuations.

The heat fluxes are modeled by

$$-\frac{\partial \overline{\rho u_i' T'}}{\partial x_i} = \kappa_t \frac{\partial \bar{T}}{\partial x_i}, \tag{4.9}$$

the extra term $xtra$ is

$$xtra = \sigma_k \frac{\partial k}{\partial x_i}, \tag{4.10}$$

with

$$\sigma_k = \mu_l + \frac{\mu_t}{\mathrm{Pr}_k}. \tag{4.11}$$

Now, the turbulent equations can be closed by defining the eddy viscosity μ_t and the turbulent kinetic energy k and a related Prandtl number Pr_k. The definitions depend on the turbulence model. The different models used are described below. Assuming here that μ_t and k are given quantities the averaged total energy is:

$$\bar{E} = \bar{e} + \frac{\bar{u}_i \bar{u}_i}{2} + k \tag{4.12}$$

with $\bar{e}$ being the internal energy. Furthermore we define the effective viscosity μ_{eff}

$$\mu_{eff} = \mu_l + \mu_t, \tag{4.13}$$

the effective thermal conductivity κ_{eff}

$$\kappa_{eff} = \kappa_l + \kappa_t, \tag{4.14}$$

the effective turbulent pressure p^*

$$p^* = \bar{p} + \frac{2}{3}\bar{\rho} k, \tag{4.15}$$

and the turbulent speed of sound c^*

$$c^* = \sqrt{\gamma \frac{p^*}{\bar{\rho}}}. \tag{4.16}$$

The averaged pressure is calculated using the equation of state

$$\bar{p} = (\gamma - 1)\bar{\rho}\bar{e} \tag{4.17}$$

and the total Enthalpy is

$$\bar{H} = \bar{E} + \frac{p^*}{\bar{\rho}}. \tag{4.18}$$

Now, we formally rewrite the fluxes, neglecting the overline bars for marking average values.

$$\vec{F}_i = \begin{pmatrix} \rho u \\ \rho u^2 + p^* \\ \rho uv \\ \rho uw \\ \rho Hu \end{pmatrix}, \quad \vec{F}_v = - \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + k_{eff}\frac{\partial T}{\partial x} - \sigma_k \frac{\partial k}{\partial x} \end{pmatrix}, \tag{4.19}$$

$$\vec{G}_i = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p^* \\ \rho vw \\ \rho Hv \end{pmatrix}, \quad \vec{G}_v = - \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{xy} + v\tau_{yy} + w\tau_{yz} + k_{eff}\frac{\partial T}{\partial y} - \sigma_k \frac{\partial k}{\partial y} \end{pmatrix}, \tag{4.20}$$

$$\vec{H}_i = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p^* \\ \rho Hw \end{pmatrix}, \quad \vec{H}_v = - \begin{pmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{xx} \\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + k_{eff}\frac{\partial T}{\partial z} - \sigma_k \frac{\partial k}{\partial z} \end{pmatrix} \tag{4.21}$$

with the stresses

$$\tau_{ij} = \mu_{eff} \cdot S_{ij}. \tag{4.22}$$

Following this notation in the code μ, κ and the pressure p are substituted by their turbulent counterparts μ_{eff}, κ_{eff} and p^*, respectively. Neglecting the turbulent contributions, i.e. $\mu_{eff} = 0$ and $k = 0$, leads again to the Navier-Stokes equations for laminar flows (also the extra term vanishes). This implementation allows in a straightforward manor to switch between the turbulent and the laminar equations.

4.2 Turbulence modeling in the DLR TAU-Code

Currently the DLR TAU-Code provides two classes of turbulence models

- RANS turbulence models
  - One-equation turbulence models based on the Spalart-Allmaras model
  - Two-equation turbulence models based on the Wilcox k-ω model
  - Explicit algebraic Reynolds stress models (EARSM) based on k-ω model
- DES models (coupling RANS in boundary layers and LES in separation regions)
  - SA-DES (DES based on Spalart-Allmaras original model)
  - MeSST-DES (DES based on Menter SST k-ω model)
  - XLES (DES based on NLR TNT k-ω model and one-equation SGS model)

4.2.1 Preliminary relations for eddy-viscosity models

We introduce the following notation for the rate-of-strain tensor

$$\mathbb{S}(\vec{u}) \;=\; \frac{1}{2}(\vec{\nabla}\vec{u} + \vec{\nabla}\vec{u}^T) - \frac{1}{3}\vec{\nabla}\cdot\vec{u}\,\mathbb{I}$$

or in componentwise notation

$$(\mathbb{S}(\vec{u}))_{ij} \;=\; \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right) \;-\; \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta ij \;\;.$$

The strain rate for compressible flows $\overline{S}$ is given by

$$\overline{S}^2 \equiv 2\mathbb{S}(\vec{u}):\mathbb{S}(\vec{u}) = \frac{1}{2}(\vec{\nabla}\vec{u}+\vec{\nabla}\vec{u}^T):(\vec{\nabla}\vec{u}+\vec{\nabla}\vec{u}^T)-\frac{2}{3}(\vec{\nabla}\cdot\vec{u})^2$$

or equivalently in componentwise notation

$$\overline{S}^2 \equiv 2\mathbb{S}(\vec{u}):\mathbb{S}(\vec{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)-\frac{2}{3}\left(\frac{\partial u_k}{\partial x_k}\right)^2$$

Using the eddy-viscosity hypothesis for compressible flows. the turbulent stress tensor can be written as

$$-\overline{\rho\vec{u}''\otimes\vec{u}''} \;=\; \mu_t\left(\vec{\nabla}\vec{u}+\vec{\nabla}\vec{u}^T-\frac{2}{3}\vec{\nabla}\cdot\vec{u}\,\mathbb{I}\right) \;-\; \frac{2}{3}\overline{\rho}k\mathbb{I}$$

or componentwise

$$\tau_{ij}\equiv-\overline{\rho u_i'' u_j''} \;=\; \mu_t\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}-\frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta ij\right) \;-\; \frac{2}{3}\overline{\rho}k\delta_{ij}$$

From this relation the production of turbulent kinetic energy becomes

$$\mathcal{P} = -\overline{\rho\vec{u}''\otimes\vec{u}''}:\vec{\nabla}\vec{u} = \mu_t\overline{S}^2-\frac{2}{3}\rho k\vec{\nabla}\cdot\vec{u}\;.$$

4.3 Spalart-Allmaras type turbulence models

The Spalart-Allmaras turbulence model is a one-equation model with a transport equation directly formulated for the eddy viscosity. The part of the reynolds-stress tensor containing the turbulent kinetic energy is neglected, i.e. $k = 0$. Currently the following SA type model are implemented

- Spalart-Allmaras model original version
- Spalart-Allmaras model with Edwards modification
- Spalart-Allmaras model modified version
- Strain-adaptive Spalart-Allmaras (SALSA) model
  (but this model is not just yet available in a release version of TAU)

In the present implementation a variable density is considered which leads to the following equation:

$$\frac{\partial\rho\tilde{\nu}}{\partial t}+\frac{\partial\rho u_i\tilde{\nu}}{\partial x_i} = P+\left(\frac{\partial}{\partial x_i}\left(\frac{\mu_l+\tilde{\mu}}{\sigma}\frac{\partial\tilde{\nu}}{\partial x_i}\right)+\rho\frac{c_{b2}}{\sigma}(\frac{\partial\tilde{\nu}}{\partial x_i})^2\right)-D \tag{4.23}$$

The terms on the RHS of the turbulence transport equation represent production, gradient diffusion and the wall destruction of the turbulent kinematic viscosity, respectively. The production is defined as

$$P = c_{b1}\rho\tilde{S}\tilde{\nu} \tag{4.24}$$

The destruction is defined as

$$D = c_{w1} f_w \rho (\frac{\tilde{\nu}}{d})^2 \tag{4.25}$$

with d being the wall-distance. $\tilde{\nu}_l$ is the laminar kinematic viscosity. ν_t is the turbulent kinematic viscosity. $\tilde{\nu}$ equals ν_t except in the buffer layer, it is modified in order to account for the wall-near viscous region. For stability reasons (especially in multigrid mode) we limit $\tilde{\nu} = MAX(\tilde{\nu}, 0)$. The eddy viscosity is defined as:

$$\mu_t = \rho \nu_t \qquad \nu_t = f_{v1} \tilde{\nu} \qquad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \qquad \chi = \frac{\tilde{\nu}}{\nu_l} \tag{4.26}$$

The definition of the production term (i.e. the definition of $\tilde{S}$) and the definition of the destruction term (i.e. the definition of the wall-blockage function f_w) depends on the version of the turbulence model considered.

4.3.1 Spalart-Allmaras model – Original version

The originally proposed version of the model by Spalart and Allmaras [66] represents the total production of turbulence by a modified magnitude of the vorticity:

$$\tilde{S} = |\vec{\omega}| + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2} \tag{4.27}$$

with

$$\vec{\omega} = \begin{pmatrix} w_y - v_z \\ u_z - w_x \\ v_x - u_y \end{pmatrix} \tag{4.28}$$

and

$$f_{v2} = 1 - \frac{\chi}{1 + \chi \{_{\sqsubseteq \infty}} \qquad \chi = \frac{\tilde{\nu}}{\nu_l} \tag{4.29}$$

The wall-blockage function needed for the formulation of the destruction term is:

$$f_w = g \left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right]^{1/6} \tag{4.30}$$

with the limiter function g:

$$g = r + c_{w2}(r^6 - r) \tag{4.31}$$

and

$$r = \frac{\tilde{\nu}}{\tilde{S} \kappa^2 d^2} \tag{4.32}$$

The f_{v2} function in this formulation can lead to numerical problems, because of a negative regime. Thus $\tilde{S}$ also can become negative. For that reason the implementation is:

$$r = \frac{\tilde{\nu}}{MAX(\tilde{S}, \varepsilon) \kappa^2 d^2} \qquad \varepsilon = 10^{-16} \tag{4.33}$$

4.3.2 Edwards version

A second version of the model considered is according to Edwards [21]. The differences relate mainly to the modeled near-wall behavior. This formulation avoids negative values of $\tilde{S}$ and is numerical more stable:

$$\tilde{S} = [\frac{1}{\chi} + f_{v1}] \quad \sqrt{\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\frac{\partial u_i}{\partial x_j} - \frac{2}{3}\left(\frac{\partial u_k}{\partial x_k}\right)^2} \tag{4.34}$$

and

$$r = tanh\left(\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}\right) \quad / \quad tanh(1.0) \tag{4.35}$$

For numerical reasons we limit $\chi = MAX(\frac{\tilde{\nu}}{\nu_l}, 10^{-16})$ and we modified the computation of the r-function using $MAX(\tilde{S}, 10^{-16})$ instead of $\tilde{S}$.

4.3.3 Initial, freestream and boundary conditions

As initial conditions we prescribe reference conditions for $\tilde{\nu}$ in the whole flow regime. The reference turbulent viscosity is an input parameter determining the ratio to the laminar kinematic viscosity $\tilde{\nu}/\nu_l$. The default is a low turbulence ($\tilde{\nu}/\nu_l << 1$), as it is expected in laminar flow. At inflow boundaries the kinematic viscosity is set according to the specific boundary parameters. The no slip wall condition for $\tilde{\nu}$ is $\tilde{\nu} = 0$.

4.3.4 Model constants

The following empirical coefficients are used in the above described equations. Note, that the constants c_{t3} and c_{t4} are according to Spalart [66].

$$\begin{aligned} &\kappa = 0.41\,, \quad c_{b1} = 0.1355\,, \quad c_{b2} = 0.622\,, \quad c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{c_{b2}+1}{\sigma}\,, \\ &c_{w2} = 0.3\,, \quad c_{w3} = 2.0\,, \quad c_{v1} = 7.1 \end{aligned} \tag{4.36}$$

4.3.5 Strain-adaptive Spalart-Allmaras (SALSA) model

The strain-adaptive Spalart-Allmaras model was proposed by Rung et al.,
see [62]. The modified production term is given by

$$\mathcal{P} \;=\; c_{b1}\,\rho\,\tilde{S}\,\tilde{\nu}\,C_\mu^*$$

with

$$\begin{aligned} C_\mu^* \;&=\; c_{b1}\sqrt{\Gamma}\,, \qquad \Gamma \;=\; \min[c_{\Gamma,1}\,,\; \max(\gamma\,,\; c_{\Gamma,2})]\,, \qquad \gamma \;=\; \max(\alpha_1\,,\; \alpha_2) \\ \alpha_1 \;&=\; \left(1.01\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}\right)^{0.65}\,, \qquad \alpha_2 \;=\; \max\Big(\,0\,,\; 1-\tanh\Big(\frac{\chi}{68}\Big)\Big)^{0.65} \end{aligned}$$

where two options for the coefficients $c_{\Gamma,1}$ and $c_{\Gamma,2}$ are available: For transonic cruise flight $c_{\Gamma,1} = 1.2$ and $c_{\Gamma,2} = 0.8$ and for subsonic high lift $c_{\Gamma,1} = 1.5$ and $c_{\Gamma,2} = 0.6$.
The modified destruction term becomes

$$D \;=\; c_{w11}\,f_w\rho\left(\frac{\tilde{\nu}}{d}\right)^2\,, \qquad c_{w11} \;=\; C_\mu^*\frac{c_{b1}}{\kappa^2} \;+\; \frac{1+c_{b2}}{\sigma}$$

The parameter r is given by

$$r = \min\left(1\ ,\ \sqrt{\frac{\rho_{0,\infty}}{\rho}}\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}\right)\ , \qquad \rho_{0,\infty} = \rho_\infty\left(1+(\gamma-1)\frac{Ma^2}{2}\right)^{\frac{1}{\gamma-1}}\ , \quad \tilde{S} = \sqrt{2tr\{\mathbb{S}^2\}}.$$

4.4 Two-equation k-ω models

Currently the following two-equation k-ω models are implemented

- Wilcox k-ω model
- Menter Baseline model
- Menter SST k-ω model
- NLR TNT k-ω model
- Wilcox SST k-ω model
- The Menter two-layer k-ε model
- LEA k-ω model

4.4.1 The Wilcox k-ω

The Wilcox k-ω model (cf. [80], p. 121 with model constants given in [80], p.129) computes μ_t from the formula

$$\mu_t = \rho\frac{k}{\omega}$$

where k and ω are the solution of

$$\begin{aligned}
\frac{\partial(\rho k)}{\partial t} + \vec{\nabla}\cdot(\vec{u}\rho k) - \vec{\nabla}\cdot\left((\mu+\sigma_k\mu_t)\,\vec{\nabla}k\right) &= \tilde{\mathcal{P}} - \beta_k\rho k\omega \\
\frac{\partial(\rho\omega)}{\partial t} + \vec{\nabla}\cdot(\vec{u}\rho\omega) - \vec{\nabla}\cdot\left((\mu+\sigma_\omega\mu_t)\,\vec{\nabla}\omega\right) &= \gamma\frac{\rho}{\mu_t}\mathcal{P} - \beta_\omega\rho\omega^2
\end{aligned}$$

or equivalently in componentwise notation

$$\begin{aligned}
\frac{\partial(\rho k)}{\partial t} + \frac{\partial}{\partial x_j}(u_j\rho k) - \frac{\partial}{\partial x_j}\left((\mu+\sigma_k\mu_t)\,\frac{\partial k}{\partial x_j}\right) &= \tilde{\mathcal{P}} - \beta_k\rho k\omega \\
\frac{\partial(\rho\omega)}{\partial t} + \frac{\partial}{\partial x_j}(u_j\rho\omega) - \frac{\partial}{\partial x_j}\left((\mu+\sigma_\omega\mu_t)\,\frac{\partial\omega}{\partial x_j}\right) &= \gamma\frac{\rho}{\mu_t}\mathcal{P} - \beta_\omega\rho\omega^2
\end{aligned}$$

where the production in the k-equation is limited by

$$\tilde{\mathcal{P}} \ = \ \min(\ \mathcal{P}\ ;\ C_{limitP_k}\varepsilon\ ) \quad \text{with} \quad \varepsilon = \beta_k\rho k\omega$$

where C_{limitP_k} is a user-chosen constant with default $C_{limitP_k} = 1000.0$ and with model constants

$$\beta_k = 0.09\ , \quad \gamma = 0.55555556\ , \quad \sigma_k = 0.5\ , \quad \sigma_\omega = 0.5\ .$$

and β_ω is calculated from the log-layer relation

$$\beta_\omega = \beta_k\left(\gamma + \frac{\sigma_\omega\kappa^2}{\sqrt{\beta_k}}\right)\ , \quad \kappa = 0.41$$

which implies $\beta_\omega = 0.07521$.

4.4.2 The Menter baseline model

The underlying idea of the baseline model proposed by [51] is a blending of the Wilcox k-ω model in the inner part of the boundary layer and the k/ε model in the outer part of the boundary layer. The aim is to obtain both the accuracy in the near-wall region of the Wilcox k-ω model and the freestream independence in the outer part of the boundary layer of the k/ε model. For this purpose, the k/ε model is transformed into a k-ω formulation. The difference between both formulations is that an additional cross-diffusion term appears in the ω equation and that the modeling coefficients are different.
In the baseline model k and ω are the solution of

$$\begin{aligned}
\frac{\partial(\rho k)}{\partial t} + \vec{\nabla}\cdot(\rho\vec{u}k) - \vec{\nabla}\cdot\left((\mu+\mu_t\sigma_k)\,\vec{\nabla}k\right) &= \tilde{\mathcal{P}} - \beta_k\rho k\omega \\
\frac{\partial(\rho\omega)}{\partial t} + \vec{\nabla}\cdot(\rho\vec{u}\omega) - \vec{\nabla}\cdot\left((\mu+\mu_t\sigma_\omega)\,\vec{\nabla}\omega\right) &= \frac{\gamma\rho}{\mu_t}\mathcal{P} - \beta_\omega\rho\omega^2 \\
&\quad + 2\sigma_{\omega 2}(1-F_1)\frac{\rho}{\omega}\vec{\nabla}k\cdot\vec{\nabla}\omega
\end{aligned}$$

The coefficients $\phi \in \{\sigma_k,\ \sigma_\omega,\ \gamma,\ \beta_\omega\}$ of the model are interpolated using the blending formula

$$\phi = F_1\phi_1 + (1-F_1)\phi_2$$

between those of the k-ω model, $\phi_1 \in \{\sigma_{k1},\ \sigma_{\omega 1},\ \gamma_1,\ \beta_{\omega 1}\}$ (inner layer), and those of the k/ε model, $\phi_2 \in \{\sigma_{k2},\ \sigma_{\omega 2},\ \gamma_2,\ \beta_{\omega 2}\}$ (outer layer)

$$\begin{aligned}
\text{Inner layer}: &\quad \sigma_{k1} = 0.5,\ \sigma_{\omega 1} = 0.5, \quad \gamma_1 = 0.555556,\ \beta_{\omega 1} = \beta_k\left(\gamma_1 + \frac{\sigma_{\omega 1}\kappa^2}{\sqrt{\beta_k}}\right) \\
\text{Outer layer}: &\quad \sigma_{k2} = 1.0,\ \sigma_{\omega 2} = 0.857,\ \gamma_2 = 0.44, \quad \beta_{\omega 2} = \beta_k\left(\gamma_2 + \frac{\sigma_{\omega 2}\kappa^2}{\sqrt{\beta_k}}\right)
\end{aligned}$$

with $\kappa = 0.41$, $\beta_k = 0.09$ and using the blending function F_1 given by

$$\begin{aligned}
F_1 = \tanh(arg_1^4)\ , \quad arg_1 &= \min\left(\max\left(\frac{\sqrt{k}}{\beta_k\omega y}; \frac{500\nu}{y^2\omega}\right)\ ;\ \frac{4\rho\sigma_{\omega 2}k}{CD_{k\omega}y^2}\right) \\
CD_{k\omega} &= \max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\vec{\nabla}k\cdot\vec{\nabla}\omega\ ;\ 10^{-20}\right)
\end{aligned}$$

Substitution gives $\beta_{\omega 1} = 0.07522$ and $\beta_{\omega 2} = 0.08282$.
The function F_1 controls the blending of the model coefficients and the cross diffusion term. Therefore F_1 has to be one in the near-wall region and in the logarithmic layer and has to tapper off to zero well within the wake region of the boundary layer in order to prevent the freestream dependence of the k-ω model.
The production limiter was devised by Menter for the following reason: In regions of small values for ω, erroneous spikes of ν_t can occur. These might stem from the term $\nu_t\omega^{-1}\mathcal{S}^2$ in the $\nu_t = k/\omega$ equation. This production term can be amplified by small values for ω. As a remedy the production term is limited.

4.4.3 The Menter SST model

The Menter SST model (see [51]) is a further improvement of the Menter baseline model. Between the two models, there are the following two differences. First, the coefficients in the

inner layer are changed to

$$\text{Inner layer}: \quad \sigma_{k1} = 0.85, \;\; \sigma_{\omega 1} = 0.5, \;\; \gamma_1 = 0.555556, \;\; \beta_{\omega 1} = \beta_k \left(\gamma_1 + \frac{\sigma_{\omega 1}\kappa^2}{\sqrt{\beta_k}} \right) .$$

Second, the so-called *shear-stress correction* is introduced:

$$\mu_t \;=\; \min\left(\frac{\rho k}{\omega} \;;\; \frac{a_1 \rho k}{\overline{\Omega} F_2} \right) \tag{4.37}$$

with

$$F_2 = \tanh(arg_2^2) \;, \;\; arg_2 = \max\left(2\frac{\sqrt{k}}{\beta_k \omega y} \;;\; \frac{500\nu}{y^2\omega} \right) \tag{4.38}$$

$$\overline{\Omega} = \sqrt{2\Omega : \Omega} \;, \quad \Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right) \tag{4.39}$$

where $a_1 = 0.31$ is the Bradshaw constant. The underlying idea is to remedy the tendency of two-equation models to overestimate the shear stress, in particular for flows with adverse pressure gradients. For this purpose a bound on the stress-intensity ratio $\overline{u'v'}/k$ is imposed. The limitation of ν_t ensures that $\overline{u'v'}/k \leq a_1$, with $a_1 = 0.31$.

4.4.4 The NLR TNT-modification of the k-ω model

The objective of the turbulent/non-turbulent (TNT) interface analysis of [38] is to solve the freestream dependency of the Wilcox k-ω model. Again $\mu_t = \rho k/\omega$ and k and ω are given by the solution of

$$\begin{aligned}
\frac{\partial(\rho k)}{\partial t} + \vec{\nabla}\cdot(\vec{u}\rho k) - \vec{\nabla}\cdot\Big((\mu + \sigma_k \mu_t)\,\vec{\nabla} k \Big) \;&=\; \tilde{\mathcal{P}} - \beta_k \rho k \omega \\
\frac{\partial(\rho \omega)}{\partial t} + \vec{\nabla}\cdot(\vec{u}\rho \omega) - \vec{\nabla}\cdot\Big((\mu + \sigma_\omega \mu_t)\,\vec{\nabla} \omega \Big) \;&=\; \gamma \frac{\omega}{k}\mathcal{P} - \beta_\omega \rho \omega^2 + C_D
\end{aligned}$$

where the cross-diffusion term is given by

$$C_D \;=\; \sigma_d \frac{\rho}{\omega} \max\Big(\vec{\nabla} k \cdot \vec{\nabla}\omega \;,\; 0 \Big) \;.$$

The model constants are given by

$$\beta_k = 0.09 \;, \quad \gamma = 0.55555556 \;, \quad \sigma_k = \frac{2}{3} \;, \quad \sigma_\omega = 0.5 \;, \quad \sigma_d = 0.5 \;.$$

and β_ω is again calculated from the log-layer relation

$$\beta_\omega = \beta_k \left(\gamma + \frac{\sigma_\omega \kappa^2}{\sqrt{\beta_k}} \right) \;, \quad \kappa = 0.41 \;.$$

4.4.5 The Wilcox k-ω model with SST correction

The Wilcox k-ω model with SST correction is based on the standard Wilcox k-ω model. It uses the same coefficients as the standard Wilcox model. Put in other words, no blending between inner and outer layer as for the Menter baseline model and the Menter SST model is used.

As a modification of the standard model, this model uses the same shear-stress correction as proposed by the Menter SST model

$$\mu_t \;=\; \min\left(\frac{\rho k}{\omega}\,;\,\frac{a_1\rho k}{\overline{\Omega}F_2}\right)$$

with

$$F_2 = \tanh(arg_2^2)\;,\quad arg_2 = \max\left(2\frac{\sqrt{k}}{\beta_k\omega y}\,;\,\frac{500\nu}{y^2\omega}\right)$$
$$\overline{\Omega} = 2\Omega:\Omega\;,\quad \Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right)$$

where $a_1 = 0.31$ is the Bradshaw constant.

4.4.6 The Menter two-layer k-ε model

The Menter two-layer k-ε model is essentially the Menter Baseline model. The only difference between both models is that the Menter two-layer k-ε model uses the following set of coefficients for inner and outer layer resp.

$$\text{Inner layer}: \quad \sigma_{k1} = 0.5,\ \sigma_{\omega 1} = 0.4,\quad \gamma_1 = 0.555556,\ \beta_{\omega 1} = \beta_k\left(\gamma_1 + \frac{\sigma_{\omega 1}\kappa^2}{\sqrt{\beta_k}}\right)$$
$$\text{Outer layer}: \quad \sigma_{k2} = 1.0,\ \sigma_{\omega 2} = 0.857,\ \gamma_2 = 0.44,\quad \beta_{\omega 2} = \beta_k\left(\gamma_2 + \frac{\sigma_{\omega 2}\kappa^2}{\sqrt{\beta_k}}\right)$$

where the modified coefficient $\sigma_{\omega 1} = 0.4$ may be motivated from the corresponding coefficient σ_ε appearing in the k-ε model after transforming the ε-equation into an equation for ω, see [80], Chapter 4.

4.4.7 LEA k-ω model by Rung

The LEA k-ω model by Rung is derived from the RQEVM by Rung (see [61]) and is described also in [24], pp.25. The eddy viscosity is computed by

$$\mu_t \;=\; \frac{C_\mu^*}{C_\mu}\rho\frac{k}{\omega}$$

where k and ω are the solution of the k-ω equations from the standard Wilcox k-ω model and

$$C_\mu^* = \frac{\beta_1}{1 - \frac{2}{3}\eta^2 + 2\xi^2}\;,\qquad C_\mu = 0.09$$

with

$$\begin{aligned}
\eta^2 &= \frac{\beta_3^2\, S^2}{8}\,, \qquad \xi^2 = \frac{\beta_2^2\, \Omega^2}{2} \\
\beta_1 &= \frac{\frac{4}{3}-C_2}{2g}\,, \qquad \beta_2 = \frac{2-C_4}{2g}\,, \qquad \beta_3 = \frac{2-C_3}{g} \\
g &= f_1(C_1-1)+\frac{S^2}{4+1.83\sqrt{0.8\Omega^2+0.2S^2}} \\
f_1 &= 1+0.95\left(1-\tanh\left(\frac{S^2}{4.6625}\right)\right) \\
C_1 &= 2.6 \quad C_2 = \max\left(0.4\,;\,\frac{1.5S^{1.7}}{17.1+1.875S^{1.7}}\right)\,, \quad C_3 = 1.25\,, \quad C_4 = 0.45 \\
S &= \frac{1}{C_\mu\omega}\sqrt{2tr\{\mathbb{S}^2\}}\,, \qquad \mathbb{S}=\frac{1}{2}(\vec{\nabla}\vec{u}+\vec{\nabla}\vec{u}^T)-\frac{1}{3}\vec{\nabla}\cdot\vec{u}\,\mathbb{I} \\
\Omega &= \frac{1}{C_\mu\omega}\sqrt{-2tr\{\Omega^2\}}\,, \qquad \Omega_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}-\frac{\partial u_j}{\partial x_i}\right)\,.
\end{aligned}$$

where $tr\{\mathbb{A}\}=\mathbb{A}_{kk}$ denotes the trace of A.

4.5 Explicit algebraic Reynolds stress models (EARSM)

4.5.1 Preliminary remarks on EARSM

In explicit algebraic Reynolds stress models (EARSM) the turbulent stress tensor $-\overline{\rho\vec{u}''\otimes\vec{u}''}$ is given by an explicit algebraic relation. From this relation the production of turbulent kinetic energy can be directly computed as

$$\mathcal{P} = -\overline{\rho\vec{u}''\otimes\vec{u}''}:\vec{\nabla}\vec{u}$$

4.5.2 The EARSM by Wallin and Johansson

In the EARSM by Wallin and Johansson (cf. [76]) the Reynolds stresses are modeled by

$$-\overline{\rho u_i'' u_j''} \;=\; 2C_\mu^{eff} k\hat{S}_{ij} \;-\; \frac{2}{3}\rho k\delta_{ij} \;-\; \rho k a_{ij}^{ex}$$

where the effective C_μ-coefficient is given by

$$C_\mu^{eff} = -\frac{1}{2}(\beta_1 + II_\Omega\beta_6)\,,$$

with extra anisotropy a_{ij}^{ex}

$$\begin{aligned}
a^{ex} =& \beta_3(\hat{\Omega}^2-\frac{1}{3}II_\Omega\mathbb{I})+\beta_4(\hat{S}\hat{\Omega}-\hat{\Omega}\hat{S}) \\
&+\beta_6(\hat{S}\hat{\Omega}^2+\hat{\Omega}^2\hat{S}-II_\Omega\hat{S}-\frac{2}{3}IV\mathbb{I})+\beta_9(\hat{\Omega}\hat{S}\hat{\Omega}^2-\hat{\Omega}^2\hat{S}\hat{\Omega})\,.
\end{aligned}$$

The normalized mean strain and rotation tensors are defined as

$$\hat{S}_{ij} = \frac{\tau}{2}\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right)-\frac{\tau}{3}\frac{\partial U_k}{\partial x_k}\delta_{ij}\,, \quad \hat{\Omega}_{ij} = \frac{\tau}{2}\left(\frac{\partial U_i}{\partial x_j}-\frac{\partial U_j}{\partial x_i}\right)\,,$$

where the turbulent timescale is defined by

$$\tau = \max\left(\frac{k}{\varepsilon},\ C_\tau\sqrt{\frac{\nu}{\varepsilon}}\right)\ , \qquad \varepsilon = \beta_k k\omega$$

From this it can be seen that the effective eddy viscosity ν_t becomes

$$\nu_t \ = \ C_\mu^{eff}\rho k\tau\ .$$

Moreover we introduce the extra stress tensor b^{ex} stemming from the extra anisotropy a^{ex}, viz.,

$$b^{ex} \ = \ \rho k a^{ex}$$

The invariants are defined by

$$II_S = tr\{\hat{S}^2\}\ , \quad II_\Omega = tr\{\hat{\Omega}^2\}\ , \quad IV = tr\{\hat{S}\hat{\Omega}^2\}\ , \quad V = tr\{\hat{S}^2\hat{\Omega}^2\}$$

where $tr\{\mathbb{A}\} = \mathbb{A}_{kk}$ denotes the trace of A and with

$$II_S^{eq} = \frac{405c_1^2}{216c_1 - 160}\ .$$

The β-coefficients are given by

$$\beta_1 = -\frac{N(2N^2 - 7II_\Omega)}{Q}\ , \quad \beta_3 = -\frac{12N^{-1}IV}{Q}\ ,$$
$$\beta_4 = -\frac{2(N^2 - 2II_\Omega)}{Q}\ , \qquad \beta_6 = -\frac{6N}{Q}\ , \qquad \beta_9 = \frac{6}{Q}\ ,$$

with the denominator

$$Q = \frac{5}{6}(N^2 - 2II_\Omega)(2N^2 - II_\Omega)\ .$$

For two-dimensional mean flows and most three-dimensional flows, it is sufficient to take $N = N_c$ where

$$N_c = \begin{cases} \frac{c_1'}{3} + (P_1 + \sqrt{P_2})^{1/3} + sign(P_1 - \sqrt{P_2})\ |P_1 - \sqrt{P_2}|^{1/3} & \text{if } P_2 \geq 0 \\ \frac{c_1'}{3} + 2(P_1^2 - P_2)^{1/6}\cos\left(\frac{1}{3}\arccos\left(\frac{P_1}{\sqrt{P_1^2 - P_2}}\right)\right) & \text{if } P_2 < 0 \end{cases}$$

where the arccos function should return an angle between 0 and π and

$$P_1 = (\frac{1}{27}(c_1')^2 + \frac{9}{20}II_S - \frac{2}{3}II_\Omega)c_1'\ , \quad P_2 = P_1^2 - \left(\frac{1}{9}(c_1')^2 + \frac{9}{10}II_S + \frac{2}{3}II_\Omega\right)^3$$

and

$$c_1' = \frac{9}{4}(c_1 - 1 - \frac{2}{3}\tau\frac{\partial U_k}{\partial x_k})\ .$$

For two-dimensional mean flow, $\hat{S}_{ij}^{2D}$ has to be replaced by the compressible two-dimensional strain rate $\hat{S}_{ij}^{2D}$ which is defined by

$$\hat{S}_{ij}^{2D} = \frac{\tau}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \frac{\tau}{2}\frac{\partial U_k}{\partial x_k}\delta_{ij}^{2D}\ , \quad \text{where} \quad \delta_{ij}^{2D} = \begin{cases} 0 & \text{if } \ i = j = 3 \\ \delta_{ij} & \text{otherwise}\ . \end{cases}$$

Alternatively, this can be written as

$$\hat{S}^{2D} \;=\; \hat{S} - \frac{\tau}{2}\vec{\nabla}\cdot\vec{U}\left(\mathbb{I}^{2D} - \frac{2}{3}\mathbb{I}\right)$$

Regarding the β-coefficients, the simplifications for two-dimensional flow are

$$\beta_1 = -\frac{6}{5}\frac{N_c}{N_c^2 - 2II_\Omega}\,, \quad \beta_4 = -\frac{6}{5}\frac{1}{N_c^2 - 2II_\Omega}\,, \quad \beta_3 = \beta_6 = \beta_9 = 0.$$

Finally, the five model constants are specified by Wallin and Johansson as

$$C_\tau = 6.0\,,\;\; c_1 = 1.8\,,\;\; B_2 = 1.8\,,\;\; C'_{y1} = \frac{2.4}{26.0}\,,\;\; C'_{y2} = \frac{0.003}{26.0}.$$

As the two-equation background model for the 2D EARSM by Wallin&Johansson, the standard Wilcox k-ω model is applied (cf. [77]).
The two-equation background model for the 3D EARSM by Wallin&Johansson is the NLR TNT k-ω model (cf. [77]). In both cases $\mathcal{P}$ is computed from the relation

$$\mathcal{P} = -\overline{\rho\vec{u}''\otimes\vec{u}''} : \vec{\nabla}\vec{u} = \mu_t\overline{S}^2 \;-\; \frac{2}{3}\rho k\vec{\nabla}\cdot\vec{u} \;-\; b^{ex} : \vec{\nabla}\vec{u}\,.$$

with

$$\overline{S}^2 \equiv 2\mathbb{S}(\vec{u}) : \mathbb{S}(\vec{u}) = \frac{1}{2}(\vec{\nabla}\vec{u} + \vec{\nabla}\vec{u}^T) : (\vec{\nabla}\vec{u} + \vec{\nabla}\vec{u}^T) - \frac{2}{3}(\vec{\nabla}\cdot\vec{u})^2\,.$$

4.5.3 The Hellsten EARSM

Hellsten devised a modified set of coefficients for the k-ω background model when used with the Wallin & Johansson EARSM as constitutive model, see [30]. The only difference between the constitutive EARSM is that Hellsten uses a modified definition of c'_1 (suggested in [76], eq. (77),(78)), viz.,

$$c'_1 \;=\; \frac{9}{4}\left[\, c_1 - 1 + C_D \max(\, 1 + \beta_1^{eq} II_S \,;\, 0\,)\,\right]$$

where

$$\beta_1^{eq} \;=\; -\frac{6}{5}\frac{N^{eq}}{(N^{eq})^2 - 2II_\Omega}\,, \qquad N^{eq} = \frac{9c_1}{4}\,, \qquad C_D \;=\; 2.2.$$

Moreover, C_μ is computed using the following relation (see [22], eq.(2.39))

$$C_\mu^{eff} \;=\; \frac{3}{5}\frac{N}{N^2 - 2II_\Omega}$$

which is valid for two- and three-dimensional mean flows and can be obtained by substituting β_1 and β_6 into the the formula for C_μ^{eff} in the Wallin & Johansson EARSM. The k-ω equations read

$$\begin{aligned}
\frac{\partial(\rho k)}{\partial t} + \vec{\nabla}\cdot(\vec{u}\rho k) - \vec{\nabla}\cdot\Big((\mu + \sigma_k\mu_t)\,\vec{\nabla}k\Big) \;&=\; \tilde{\mathcal{P}} - \beta_k\rho k\omega \\
\frac{\partial(\rho\omega)}{\partial t} + \vec{\nabla}\cdot(\vec{u}\rho\omega) - \vec{\nabla}\cdot\Big((\mu + \sigma_\omega\mu_t)\,\vec{\nabla}\omega\Big) \;&=\; \gamma\frac{\omega}{k}\mathcal{P} - \beta_\omega\rho\omega^2 + C_D
\end{aligned}$$

where $\mathcal{P}$ is computed as in the Wallin&Johansson EARSM and the cross-diffusion term is given by

$$C_D \;=\; \sigma_d\frac{\rho}{\omega}\max\Big(\,\vec{\nabla}k\cdot\vec{\nabla}\omega\,;\,0\,\Big)\,.$$

The coefficients $\phi \in \{\sigma_k,\ \sigma_\omega,\ \gamma,\ \beta_\omega,\ \sigma_d\ ,\kappa\}$ of the model are interpolated using the blending formula

$$\phi = f_{mix}\phi_1 + (1 - f_{mix})\phi_2$$

between the set of inner layer coefficients $\phi_1 \in \{\sigma_{k1},\ \sigma_{\omega 1},\ \gamma_1,\ \beta_{\omega 1}\ \sigma_{d1}\ ,\kappa_1\}$, and the set of outer layer coefficients $\phi_2 \in \{\sigma_{k2},\ \sigma_{\omega 2},\ \gamma_2,\ \beta_{\omega 2}\ \sigma_{d2}\ ,\kappa_2\}$

$$\begin{aligned} \text{Inner layer}: &\quad \sigma_{k1} = 1.1, \quad \sigma_{\omega 1} = 0.53, \quad \gamma_1 = 0.518, \quad \sigma_{d1} = 1.0\ , \quad \kappa_1 = 0.42 \\ \text{Outer layer}: &\quad \sigma_{k2} = 1.1, \quad \sigma_{\omega 2} = 1.00, \quad \gamma_2 = 0.440, \quad \sigma_{d2} = 0.4\ , \quad \kappa_2 = 0.3795 \end{aligned}$$

and

$$\beta_{\omega 1} = \beta_k \left(\gamma_1 + \frac{\sigma_{\omega 1}\kappa^2}{\sqrt{\beta_k}} \right)\ , \qquad \beta_{\omega 2} = 0.0828\ , \qquad \beta_k = 0.09$$

The mixing function f_{mix} is based on the idea of Menters BSL model. It returns a value equal to one almost up to the edge of the boundary layer and tappers off quickly at the outer edge and equals zero in the free stream and in laminar regions. Hellsten proposed to set

$$f_{mix} = \tanh(C_{mix}\Gamma^4)\ , \qquad C_{mix} = 1.5\ , \qquad \Gamma = \min[\ \max(\ \Gamma_1\ ;\ \Gamma_2\ )\ ;\ \Gamma_3\ ]$$

where

$$\Gamma_1 = \frac{\sqrt{k}}{\beta^* \omega d}\ , \qquad \Gamma_2 = \frac{500\mu}{\rho\omega d^2}\ , \qquad \Gamma_3 = \frac{20k}{\max[d^2(\vec{\nabla}k \cdot \vec{\nabla}\omega)/\omega\ ;\ 200k_\infty]}$$

with d being the distance to the closest wall and k_∞ denoting the user-specified free-stream value of k.

4.5.4 RQEVM k-ω model by Rung

The RQEVM k-ω model by Rung is described in [61]. The Reynolds stresses are modeled by

$$-\overline{\rho u_i'' u_j''} \ = \ -2\rho k b_{ij} \ - \ \frac{2}{3}\rho k \delta_{ij}$$

where the anisotropy tensor of the Reynolds stresses b_{ij} is given by the formula

$$b_{ij} \ = \ -\frac{\nu_t}{k}\left[\ \mathbb{S}_{ij} + \beta_2 \frac{k}{\varepsilon}\left(\ \mathbb{S}_{ik}\Omega_{kj} + \mathbb{S}_{jk}\Omega_{ki}\ \right) - \beta_3 \frac{k}{\varepsilon}\left(\ \mathbb{S}_{ij}^2 - \frac{1}{3}\delta_{ij}\mathbb{S}_{kk}^2\ \right)\ \right]$$

with $\varepsilon = C_\mu k\omega$ and $\mathbb{A}_{ij}^2 = \mathbb{A}_{ik}\mathbb{A}_{kj}$, $\mathbb{A}_{jj}^2 = \mathbb{A}_{jk}\mathbb{A}_{kj}$ for some tensor $\mathbb{A}$. The mean strain and rotation tensors are defined as

$$\mathbb{S}_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \frac{1}{3}\frac{\partial U_k}{\partial x_k}\delta_{ij}\ , \quad \Omega_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i}\right)\ .$$

The effective eddy viscosity is computed by

$$\mu_t \ = \ \rho\nu_t\ , \qquad \nu_t \ = \ C_\mu^* \frac{k^2}{\varepsilon}\ , \qquad C_\mu = 0.09\ , \qquad \varepsilon = C_\mu k\omega$$

and

$$C_\mu^* = \frac{\beta_1}{1 - \frac{2}{3}\eta^2 + 2\xi^2}$$

with

$$\begin{aligned}
\eta^2 &= \frac{\beta_3^2\, S^2}{8}\,, \qquad \xi^2 = \frac{\beta_2^2\,\Omega^2}{2} \\
\beta_1 &= \frac{\frac{4}{3}-C_2}{2g}\,, \qquad \beta_2 = \frac{2-C_4}{2g}\,, \qquad \beta_3 = \frac{2-C_3}{g} \\
g &= f_1(C_1-1)+\frac{S^2}{4+1.83\sqrt{0.8\Omega^2+0.2S^2}} \\
f_1 &= 1+0.95\left(1-\tanh\left(\frac{S^2}{4.6625}\right)\right) \\
C_1 &= 2.6 \quad C_2 = \max\left(0.4\,;\,\frac{1.5S^{1.7}}{17.1+1.875S^{1.7}}\right)\,, \quad C_3 = 1.25\,, \quad C_4 = 0.45 \\
S &= \frac{1}{C_\mu\omega}\sqrt{2tr\{\mathbb{S}^2\}}\,, \qquad \mathbb{S}=\frac{1}{2}(\vec{\nabla}\vec{u}+\vec{\nabla}\vec{u}^T)-\frac{1}{3}\vec{\nabla}\cdot\vec{u}\,\mathbb{I} \\
\Omega &= \frac{1}{C_\mu\omega}\sqrt{-2tr\{\Omega^2\}}\,, \qquad \Omega_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}-\frac{\partial u_j}{\partial x_i}\right)\,.
\end{aligned}$$

where $tr\{\mathbb{A}\}=\mathbb{A}_{kk}$ denotes the trace of A.
The background two-equation model is the standard Wilcox k-ω model where $\mathcal{P}$ is computed from the relation

$$\mathcal{P} = -\overline{\rho\vec{u}''\otimes\vec{u}''} : \vec{\nabla}\vec{u}\,.$$

4.6 Vortical Correction Models

While advances have been made in second moment closure and LES/DES methods, a significant majority of computational fluid modeling in industry is still undertaken with simpler one- and two-equation Reynolds Averaged Navier-Stokes (RANS) closures. An important ingredient in this class of model is the adoption of the linear Boussinesq stress-strain hypothesis in linking mean velocity field gradients to the fluid stress. There are well-known deficiencies of this hypothesis (e.g. see Chapter 6, "Turbulence Modeling for CFD", Wilcox. D.C.), particularly with respect to flows which experience mild to severe streamline curvature influences, adverse pressure gradients, system rotation, and three-dimensional effects. It is useful to require that the system equations are invariant upon a change of reference frame or coordinate system. This requires that the form of the equation remains unchanged, so that no additional source/sink terms appears as a consequence of the transformation. Equations which display this property are termed as **invariant**, and it is desirable that model equations are invariant upon transformation from one reference frame to another. The simplest form of invariance is known as Galilean invariance (GI) where the equations are identical in any two inertial reference frames (frames which move with a constant relative velocity). In general, transformations to rotating coordinate systems cannot be done in a frame invariant manner and additional source/sink terms (related to additional acceleration components) will appear in the model equations. These additional terms act to suppress or magnify the production of turbulence stresses. A demonstration is afforded in the following thought experiment, the motivation of which comes from Durbin (Section 7.2.2) [Durbin, P.A., "Statistical theory and modeling for Turbulent Flows", Wiley and Sons, 2000]. The geometry of the experiment is illustrated in fig. (4.1). A cylindrical coordinate basis is chosen for the coordinate system, with unit normals e_1 and e_2 illustrated in the figure. For simplicity only a two-dimensional flow is chosen, so that the coordinates are given by $x_1 = R\Phi$, and $x_2 = r$. R is a constant in the following discussion. Assuming that the flow remains attached

to the body, then in the vicinity of the wall the velocity vector is given by $U(x_1,x_2) = \mathbf{U}e_1$. The arclength dS traveled by a fluid element is given by $dS = Rd\Phi$. Differentiating both left and right side with respect to time, the angular velocity of the fluid element is written as $\Omega = \pm U/R$, with sign being determined by the direction of the angular velocity (A right handed coordinate system is assumed).

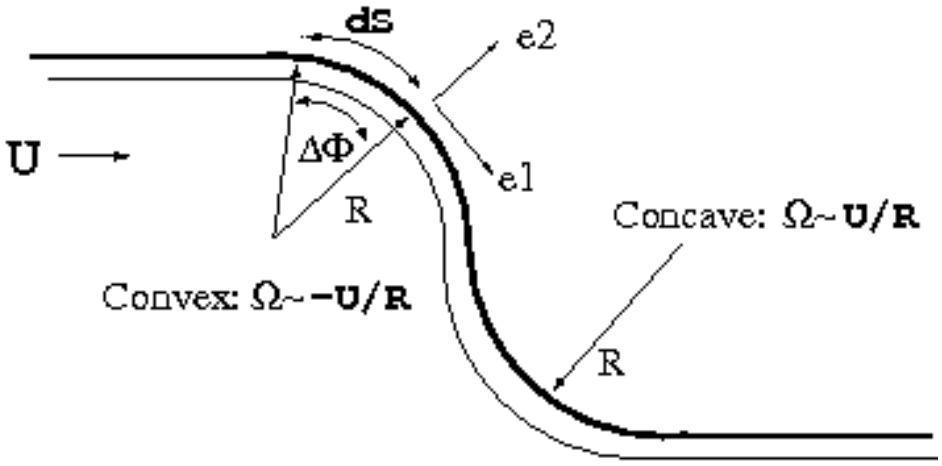


Figure 4.1: Attached flow over both convex and concave surfaces.

The components for the velocity gradient tensor are given by

$$U_{i,j} = \begin{pmatrix} 0 & -\frac{U(r)}{R} & 0 \\ \frac{\partial U(r)}{\partial r} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.40}$$

Production of shear generated turbulence is defined as

$$P_{ij} = -\overline{u_i u_j}\frac{\partial U_i}{\partial x_j}, \tag{4.41}$$

with components being given by

$$P_{ij} = \begin{pmatrix} -2\overline{uv}\frac{\partial U(r)}{\partial r} & -\overline{v^2}\frac{\partial U(r)}{\partial r} + \overline{u^2}\frac{U}{R} & 0 \\ -\overline{v^2}\frac{\partial U(r)}{\partial r} + \overline{u^2}\frac{U}{R} & 2\overline{uv}\frac{U}{R} & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.42}$$

The kinetic energy is conventionally defined as

$$\frac{1}{2}(P_{11} + P_{22}) = -\overline{uv}\left(\frac{\partial U}{\partial r} - \frac{U}{R}\right). \tag{4.43}$$

Using the terms of the production tensor allows an estimate of curvature influence on the turbulence levels. Along a convex wall the velocity will increase in the radial direction so that $\frac{\partial U(r)}{\partial r} \geq 0.0$. Note that the two terms within the brackets off the right-hand side for the turbulent kinetic energy are opposite signed, so that the influence of convex curvature is to decrease the production of kinetic energy. For a concave wall, the radial momentum component decreases so that both terms are negative and total production is then enhanced. The influence of curvature

will increase as R is decreased, and from Bradshaw it is shown that the relative magnitude of curvature and shear can be estimated as

$$\frac{U/R}{\frac{\partial U(r)}{\partial r}} \sim \frac{\delta_* U(r)}{R u_*}, \tag{4.44}$$

where δ^* is the displacement thickness, and u^* is the friction velocity. Curvature effects become important when $\frac{\delta^*}{R} \geq \frac{u^*}{U}$. From Bradshaw (p-164), if the radius of curvature is less than that about twenty times the displacement thickness, then curvature will exert a significant influence. If an eddy viscosity formulation is now adopted, so that

$$\overline{uv} = \nu_T \frac{\partial U(r)}{\partial r}, \tag{4.45}$$

then $P_11 \propto \frac{\partial U(r)}{\partial r}^2 > 0$ so that the production term P_{11} will be overestimated in a convex flow. $P_{22} \propto -U\frac{\partial U(r)}{\partial r}/R$ and in a concave flow U/R and $\frac{\partial U(r)}{\partial r}$ are of opposite sign and $P_{22} > 0$, amplifying the wall normal component. The normal stress is responsible for the shear production but, from eqn. (4.42), because $\overline{v^2}$ enters the expression for P_{22} with a negative sign an increase in $\overline{v^2}$ along a concave surface will increase the magnitude of $\overline{uv}$. The influence of curvature can also be written as

$$\frac{\delta_*}{R} \geq \frac{u_* \overline{v^2}}{U \overline{u^2}}. \tag{4.46}$$

Since the ratio of $\overline{v^2}/\overline{u^2}$ is less than one in the vicinity of a wall (influence of wall-damping on the normal velocity component), the influence of curvature plays a more significant role in the shear stress $\overline{uv}$ than in the turbulent energy equation. It can also be shown that a rotation Ω contributes a term $2\Omega\left(\overline{u^2} - \overline{v^2}\right)$ to the shear stress budget. Near a wall, where $\overline{u^2} > \overline{v^2}$, the shear production is enhanced for a positive rotation and reduced when the sign of the rotation changes. These example illustrates how the simple Boussinesq closure for the turbulence stresses can over- or under-predict the level of turbulent kinetic energy, depending on the nature of the flow. As noted by Spalart and Schur[1996], thin shear flows with weak rotation/curvature and homogenous rotating shear flow are reasonably understood. In the first case there is a significant influence on the levels of turbulent shear stress, whilst for the other case a strong rotation reduces the levels of turbulent stress. Additional, the strong rotation found in a free vortex core has a significant influence on levels of turbulent kinetic energy within the vortex core.

Simple Frame Invariant models

In this section, reliance is placed upon the notation developed by Einstein and others in the treatment of tensor fields. As suggested by the example shown in the previous section, the failure of Reynolds Averaged Navier-Stokes solutions in the presence of coordinate system rotation and streamline curvature are due to deficiencies of the linear Boussinesq stress-strain hypothesis. Principally, the problem is that of a proper turbulent flow description within a semi-empirical modeling framework! Enhancement and suppression of turbulent production cannot be accurately modeled by conventional turbulence closures, and it is difficult to describe system rotation/curvature effects in an invariant manner. Knight and Saffman [1978] proposed using the gradient of the angle between the main axes of the the strain rate tensor and the inertial coordinate system with respect to time. Generally, earlier studies proposed that the equations could be sensitized to rotation by using non-invariant expressions of the form U/R where U

is a scale velocity and R is the streamline curvature radius. Some models of this nature will be discussed shortly, but first models which display frame invariance for simple situations are described. Influenced by the ideas of Knight and Saffman, Spalart and Schur [1977] proposed the following sensor for system and streamline curvature

$$E^* = \frac{S_{jk}}{S_{mn}S_{mn}} \left(\omega_{ik} + \varepsilon_{ijk}\Omega_j\right) \frac{DS_{ij}}{Dt}, \tag{4.47}$$

where $S_{ij} = 0.5 * \left(u_{i,j} + u_{j,i}\right)$ is the symmetric part of the strain rate tensor, and $\omega_{ik} = 0.5 * \left(u_{i,j} - u_{j,i}\right)$ is the anti-symmetric part of this tensor. The symbol ε_{pqr} is the permutation symbol which has the character of an oriented Cartesian tensor. Ω_j is defined as the rotation vector of the coordinate system. The Lagrangian (or convective) derivative of the symmetric strain rate contribution is not a tensor, so that the above expression is only invariant for simple cases. The production term of the Spalart-Allmaras eddy viscosity transport equation is given by $P = c_{b1}\tilde{S}\tilde{\nu}_T$, and in order to mimic the influence of rotation/curvature the production term can be modified by a rotation function $P^* = P f_{r1}$ where $f_{r1}\left(r^*, \tilde{r}\right)$ which is a function of rotation/curvature sensors r^* and $\tilde{r}$. For the SARC model, the rotation function is given by the following equation

$$f_{r1}(r^*, \tilde{r}) = (1 + c_{r1}) \frac{2 * r^*}{1 + r^*} \left[1 - c_{r3} \tan^{-1}\left(c_{r2}\tilde{r}\right)\right] - c_{r1}. \tag{4.48}$$

Note that Ω_n is a component of the reference frame angular velocity. The quantities r^* and $\tilde{r}$ are defined as

$$r^* = S/\omega, \tag{4.49}$$

and (4.50)

$$\tilde{r} = 2\omega_{ik}S_{jk}\left[\frac{DS_{ij}}{dT} + \left(\varepsilon_{imn}S_{jn} + \varepsilon_{jmn}S_{in}\right)\Omega_m\right]/D^4. \tag{4.51}$$

The usual definitions for S_{ij} and ω_{gh} apply whereby they are the symmetric and anti-symmetric part of the stress tensor respectively (with an additional contribution from system rotation):

$$S_{ij} = 0.5 * \left(u_{i,j} + u_{j,i}\right), \tag{4.52}$$

and (4.53)

$$\omega_{gh} = 0.5 * \left[\left(u_{g,h} - u_{h,g}\right) + 2\varepsilon_{mhg}\Omega_m\right]. \tag{4.54}$$

Finally, $S^2 = 2S_{ij}S_{ji}$, $\omega^2 = 2\omega_{ij}\omega_{ij}$ and $D^2 = 0.5\left(S^2 + \omega^2\right)$.

The form of the equation for f_{r1} is chosen to ensure that, as the strain tensor vanishes, the function becomes insensitive to $\tilde{r}$. For thin shear flows without curvature $\tilde{r} = 0$ and $r^* = 1$, so that $f_{r1} = 1.0$, thus returning the original eddy viscosity production term for the Spalart-Allmaras model. The original coefficient are set as default values within the TAU-Code, and these have been based on the wingtip calculations of Daccles-Mariani et al. for which $c_{r1} = 1.0$ has been recommended. Tests based on curved and rotating boundary layers were also used (by Spalart and Schur) to tune $c_{r2} = 12$ and $c_{r3} = 1.0$. The two coefficients c_{r1} and c_{r3} can be modified by the user via the parameter file (see User Guide). Spalart and Schur note that an inverse tan function was chosen to allow a large slope of $\tilde{r}$ without requiring large values of $\tilde{r}$, however this comes at a cost. A smoother should always be applied to the field of computed f_{r1} for the SARC and SSARC models, since small differences in the argument to the inverse tan function are magnified by the function and act to destroy the smoothness of the computed vortical correction field.

Restricted Further simplification of the SARC models

Under special conditions, further simplification of the SARC model can be undertaken. For convenience we write eqn.(4.48) in the form presented by Kozlov et al. [Distinctive Features of the Turbulent Flow in a Trailing Vortex, "Fluid Dynamics, Vol. 39, 1, 2004, pp 69.75].

$$F = \frac{2(1+c_{r1})(R+c_{r4})}{(1+c_{r4})(R+1)}\left[1-c_{r3i}\arctan\left(c_{r2}E^*\right)\right]-c_{r1} \tag{4.55}$$

Here $R^2 = S_{ij}S_{ij}/(\omega_{nm}\omega_{nm})$ and C_4 is set to zero to recover eqn. (4.48). In the absence of rotation or streamline curvature effects in simple shear flows then $E^* = 0$ ($\tilde{r}$ in eqn. (4.48) and R = 1, so that the original SA model is recovered. However, at small values of streamline curvature then $E^* << 1$ and $R \approx 1$ so that the **arctan** function can be expanded as a power series and the following form returned:

$$F \approx 1-(1+c_{r1})\,c_{r2}c_{r3}E\,. \tag{4.56}$$

The use of this form of the rotation correction function is appropriate, for example, in an attached boundary layer on a slightly curved surface. In this case the influence of curvature is expressed through the parameters $c_{r1}..c_{r3}$. Kozlov et al. propose the following values for the coefficient set

$$c_{r1} = 0.09\left(1+0.06\frac{\nu_t}{\nu}\right) \tag{4.57}$$

$$c_{r2} = \frac{24}{(1+c_{r1})\,c_{r3}} \tag{4.58}$$

$$c_{r3} = 0.55 \tag{4.59}$$

$$c_{r4} = 0.02 \tag{4.60}$$

It is this form of the SARC model that is activated when the parameter "Kozlov modification" is set to one.

Additional comment for the SARC model

Recalling eqn. (4.51), an estimate of the spatial gradients of the strain field is required in order to compute f_r for the SARC correction. At the present time, a measure of the second spatial derivative of the velocity field is estimated as

$$S_{ij,k} = D(S_{ij}) \tag{4.61}$$

where $D()$ is the standard gradient operator in TAU. At present $D()$ is can be either a Green-Gauss or a least-squares based algorithm. On structured grids with uniform grid spacing, the exact gradient can be written in terms of the gradient operator $D()$ as

$$\frac{d\phi}{dx_j} = D(\phi)+O(\triangle_j^2), \tag{4.62}$$

for both the Green-Gauss and the Least-squares operators. However, it can be argued that the computation of second-order gradients (as required for the gradients of the strain S_{ij}) is inconsistent in that the magnitude of the numerical error cannot be guaranteed to be bounded below the value of the second-order gradient. Additionally, the order of $D()$ is reduced by one order in the near boundary regions. Studies by Lele (AIAA 89-0374 (unpublished)) suggest that if

the approximation order is one less than the order of the numerical scheme, the overall accuracy of the scheme should not be damaged so this latter consideration may not be of a serious nature. In cases where the flow field is not sufficiently smooth, problems might be observed in the application of f_r with the least-squares gradient reconstruction and it is recommended that the user switch back to the Green-Gauss formulation. The SSARC model is based upon an unpublished simplification of the SARC model by Spalart and co-workers, and so it cannot be discussed in detail in this document. Suffice to say that for steady state one-dimensional and two-dimensional flows under moderate rotation/curvature the SSARC model returns results similar to the more complex SARC model and appears to be useful. However, at high curvature and rotation rates the use of this model will result in qualitatively incorrect behavior. Thus the user is warned that this model should only be used for steady state two-dimensional problems with low to moderate rotation/curvature influences. In order to determine what constitutes an moderate curvature, the user is directed to Bradshaw (p-164), where it is qualitatively shown that if the radius of curvature is than that about twenty times the displacement thickness, then curvature will exert a significant influence. To date the behavior of the SSARC model has been as expected in simple steady one and two dimensional flows, and the user might choose to use the SSARC model in preference to the SARC model under these circumstances. Additional computational costs arise only for the SARC model. since the gradients of the stress tensor must be evaluated. Additionally developers are warned that the treatment of the block rotational velocities is relevant for the SARC model, since the block rotational velocities are directly included inside the SARC formulation.

Simple Non-Invariant models

This class of models have been developed specifically to ensure that the production of turbulence is reduced in free vortex cores for eddy viscosity turbulence models. If the production of turbulence within the vortex core is not reduced, then the turbulent eddy viscosity is over-predicted in this region which leads to rapid dissipation of the vortex. The **flower** variant again modifies the production term of the Spalart-Allmaras eddy viscosity transport equation. Using the notation adopted in the previous section, the function f_{r1}) is now only a function of r^* and is written as follows:

$$f_{r1}(r^*) = 1 + c_{rf} \min(0, r^* - 1) \tag{4.63}$$

so that when $S >> \omega \rightarrow$ the production modifier $f_{r1} = 1$, and is decreased when vorticity dominates the strain. The coefficient c_{rf} has been tuned against some wing-tip vortical flows and has a default value of 4. In order to improve the behavior of the $k-\omega$ solutions in vortex dominated flows, Kok et al. [2001] have considered two modifications of the source terms as a function of r^*. The first modification suggested was an extension of the existing k-equation production term limiter, still using the dissipation term as a limiter but now choosing the limiter coefficient as a linear function of r^*. The form of this limitation is as follows:

$$P_k = \min\left(P_k, [C_{k1} + C_{k2} \min(0, r^* - 1)]\, \rho\varepsilon\right),$$

where the two model coefficients C_{k1} and C_{k2} are greater that zero. With this limitation, the production of turbulence kinetic energy inside a vortex core is reduced and it may even be turned into a dissipation term. However, for non-equilibrium boundary-layer flows, a value of C_{k1} close to unity can result in an unphysical reduction of the turbulence production. Kok et al. have suggested setting $C_{k1} = 2.0$ and $C_{k2} = 2.0$ in equilibrium boundary layers and 8.0 elsewhere. In TAU this particular modification has been implemented. Before discussing the Kok modification that is currently implemented in the TAU-Code, it is useful to write out the production terms that are found in the k and $\omega-$ equations. Recall that the eddy viscosity is

written as

$$\nu_T = \alpha^* \frac{\rho k}{\omega}.$$

Note that in the standard $k-\omega$ model the coefficient a^* is equal to one. The production terms for the $k-\omega$ model are then written as

$$P_k = \tau_{ij} u_{ij} = \nu_t S^2 - \frac{2}{3}\rho k u_{i,i} \tag{4.64}$$

$$P_\omega = \alpha \frac{\omega}{k}\tau_{ij} = \frac{\alpha\omega}{k} P_k = \alpha\alpha^*\rho S^2 - \frac{2}{3}\alpha\rho\omega u_{i,i} \tag{4.65}$$

The modification of Kok et al. that has been implemented inside the TAU-Code leaves the production term of the $k-$equation unchanged, and modifies the production term of the $\omega-$equation instead.

$$P_\omega = \alpha\alpha^*\rho \max\left(\omega^2, S^2\right).$$

Note that the contribution associated with the dilation has been ignored. Dividing and multiplying the above equation by ω^2 enables the component of the $\omega-$equation production source term to be written as

$$P_\omega = \alpha\alpha^*\rho\omega^2 \max\left(r^{*^2}, 1\right) \tag{4.66}$$

The production of turbulent dissipation can then be written as a function of f_{r1},

$$\begin{aligned} P_\omega &= \alpha\alpha^*\omega^2 f_{r1}(r^*) \\ &\text{where} \\ f_{r1}(r^*) &= \max(r^{*^2}, 1). \end{aligned}$$

In regions of the flow where $r^* <= 1$, for example in an attached boundary layer, the production term remains unchanged. However, when the vorticity magnitude exceeds the strain magnitude the production of the dissipation is increased. This has the effect of reducing the dissipation of the turbulence kinetic energy and thereby lowering the levels of modeled eddy viscosity. There are no tuning coefficients associated with this model.

4.7 Detached-eddy simulation

Detached-eddy simulation (DES), first devised in [67], provides a single non-zonal formulation for using a RANS model in boundary layers and to switch to LES mode in regions of massive separation.

4.7.1 Spalart-Allmaras original model with DES modification

The Spalart-Allmaras original model with DES modification [67] is based on the one-equation model by Spalart-Allmaras model (see [66]) for the eddy viscosity.

$$\frac{\partial}{\partial t}(\rho\tilde{\nu}) + \vec{u}\cdot\vec{\nabla}(\rho\tilde{\nu}) - \vec{\nabla}\cdot\left(\frac{\mu_l + \rho\tilde{\nu}}{\sigma}\vec{\nabla}\tilde{\nu}\right) - \rho\frac{c_{b2}}{\sigma}(\vec{\nabla}\tilde{\nu})^2 = \mathcal{P}_\nu - \varepsilon_\nu$$

where μ_l is the laminar viscosity, and production term $\mathcal{P}_\nu$ and destruction term ε_ν being defined as

$$\mathcal{P}_\nu = c_{b1}\rho\tilde{\mathcal{S}}\tilde{\nu}\,, \qquad \varepsilon_\nu = c_{w1} f_w \rho\left(\frac{\tilde{\nu}}{\tilde{d}}\right)^2$$

This is exactly the original SA-model, except that the length scale d resp. $\tilde{d}$ in the destruction term is modified. In the SA-model, d is the distance to the nearest wall. In the DES model, d is replaced with $\tilde{d}$, which is defined as

$$\tilde{d} \;=\; \min(\, d \;,\; C_{DES}\Delta) \qquad \text{with} \quad \Delta = \max(\Delta x, \Delta y, \Delta z)$$

where $\Delta x, \Delta y, \Delta z$ denote the grid spacing in x-, y-, and z-direction resp. Defining $\Delta = \max(\Delta x, \Delta y, \Delta z)$ ensures RANS behavior in boundary layers as $d \ll \Delta$, although $\Delta y \ll d$ and the ratio between $(\Delta x \Delta y \Delta z)^{1/3}$ and d is unclear.
Calibration of the LES-mode for decaying homogeneous isotropic turbulence suggests the value $C_{DES} = 0.65$.
It can be seen from balancing production and destruction term $\mathcal{P}_\nu = \varepsilon_\nu$ that the model then reduces formally to $\tilde{\nu}\tilde{\mathcal{S}} \sim \left(\frac{\tilde{\nu}}{\Delta}\right)^2$ which is the Smagorinsky model $\nu_{SGS} \sim \tilde{\mathcal{S}}\Delta^2$.
The remainder of the model is defined as the original SA-model.
Moreover, the low-Re modification of the DES length-scale by Strelets is implemented

$$\tilde{d} \;=\; \min(\, d \;,\; \Psi C_{DES}\Delta) \qquad \text{with} \qquad \Psi^2 = \frac{1 - c_{b1} f_{v2}/(c_{w1}\kappa^2 f_w^\star)}{f_{v1}}$$

with $f_w^\star = 0.427$. The original formulation by Strelets includes the parameter f_{t2} accounting for laminar-turbulent transition, but this version is currently not implemented.
The aim of this modification is to disable the low-Re (near-wall damping) terms of the SA-model in the LES-mode in the near-wall region of a low-Re number flow, which are not in agreement with the desired Smagorinsky like SGS model behavior. Moreover, this formula ensures that the damping terms are active in RANS mode and that the modification does not lead to a discontinuity of ν_t at the RANS/LES interface.

4.7.2 Menter SST k-ω model with DES modification

A formulation of detached-eddy simulation based on the Menter SST k-ω RANS model [51] was devised in [69].
The DES length scale is defined as

$$\tilde{l} \;=\; \min(l_{k-\omega}, C_{DES}\Delta) \qquad l_{k-\omega} \;=\; \frac{k^{1/2}}{\beta_k \omega} \tag{4.67}$$

where $l_{k-\omega}$ denotes the length scale from the k-ω model and Δ is given as in the SA-DES model.
The DES modification of the k-ω model in [69] is based on the idea to keep the formulation as simple as possible with the only restriction that at equilibrium the resulting subgrid model should reduce to a Smagorinsky-like model $\nu_t \sim \mathcal{S}\Delta^2$.
Following this idea, the only term modified is the dissipative term of the k-transport equation:

$$\varepsilon \;\equiv\; D^k_{RANS} \;=\; \rho\beta_k k\omega \;=\; \rho\frac{k^{3/2}}{l_{k-\omega}} \tag{4.68}$$

Then the DES modification consists in replacing l with $\tilde{l}$ from 4.67, resulting in

$$\varepsilon \;\equiv\; D^k_{DES} \;=\; \rho\frac{k^{3/2}}{\tilde{l}} \tag{4.69}$$

The constant C_{DES} is obtained in two steps. First, for both the k-ω and the k-ε branch separated calibrations of C_{DES} in decaying homogeneous isotropic turbulence are performed. Then, the two values obtained are blended using Menter's blending function F_1

$$C_{DES} \;=\; (1 - F_1)C^{k-\varepsilon}_{DES} \;+\; F_1 C^{k-\omega}_{DES}, \quad \text{with} \quad C^{k-\varepsilon}_{DES} = 0.61\;, \quad C^{k-\omega}_{DES} = 0.78\;.$$

4.7.3 Extra-large eddy simulation XLES

The XLES model [39] can be described at follows:

- In the RANS formulation, the TNT k-ω model [38] is used.
- In the LES formulation, the k-equation SGS model is employed. Therein, k is given by the solution of the k-equation of the TNT k-ω model with modified dissipation term (see below).
- The SGS filter width, i.e., the length scale Δ in LES mode, is defined problem-specific but independent of the grid (explicit filtering).

The difference between RANS mode and LES mode lies in the modeling of the eddy viscosity and the dissipation term in the k-equation, for which different length scales are used:

Mode	eddy viscosity	length scale	dissipation
RANS	$\nu_t = l\sqrt{k}$	$l = \sqrt{k}/\omega$	$\varepsilon = \beta_k k^{3/2} l^{-1}$
LES	$\nu_t = l\sqrt{k}$	$l = C_1\Delta$ (grid-independent)	$\varepsilon = C_2 k^{3/2}\Delta^{-1}$

The composition of the RANS and LES models is obtained by replacing the length scales in the eddy viscosity and dissipation formulation by a composite length scale $\tilde{l}$

$$\tilde{l} \;=\; \min(\, l \,,\, C_1\Delta \,)\,, \quad \nu_t \;=\; \tilde{l}\sqrt{k}\,, \quad \varepsilon \;=\; \beta_k \frac{k^{3/2}}{\tilde{l}}$$

which can be rewritten as

$$\nu_t \;=\; \min\left(\frac{k}{\omega} \,,\, C_1\Delta\sqrt{k} \right)$$
$$\varepsilon \;=\; \max\left(\beta_k \omega k \,,\, C_2 \frac{k^{3/2}}{\Delta} \right)$$

with $C_2 = \beta_k / C_1$ in order to make ν_t and ε switch simultaneously.
For the k-ω model, the TNT set of coefficients is employed. The coefficient C_1 has been calibrated to $C_1 = 0.06$ for decaying homogeneous isotropic turbulence.

4.8 Scale-Adaptive Simulation Models

The Scale-Adaptive Simulation (SAS) concept applied to a RANS model is based on the introduction of the von Karman length scale into the turbulence scale equation. The information provided by the von Karman length scale allows SAS models to dynamically adjust to resolved structures in a RANS simulation, which results in a LES-like behavior in unsteady regions of the flowfield. At the same time, the model provides standard RANS capabilities in stable flow regions.
The SAS modification consists of an additional term which is added to the right hand side of the ω-equation of the SST model

$$Q_{\mathrm{SST-SAS}} \;=\; \rho F_{\mathrm{SAS}} \cdot \max\left[\tilde{\zeta}_2 \kappa \mathcal{S}^2 \frac{L}{L_{\mathrm{vK}}} - \frac{2k}{\sigma_\phi} \cdot \max\left(\frac{|\vec{\nabla} k|^2}{k^2}, \frac{|\vec{\nabla}\omega|^2}{\omega^2} \right),\, 0 \right]$$

with van Karman length scale L_{vK} given by

$$L_{\mathrm{vK}} = \kappa\frac{\mathcal{S}}{U''}, \qquad U'' = ||(\vec{\nabla}^2 u_1, \vec{\nabla}^2 u_2, \vec{\nabla}^2 u_3)^T|| = \sqrt{\sum_{i=1}^{d}\left(\sum_{j=1}^{d}\frac{\partial^2 u_i}{\partial x_j^2}\right)^2}$$

and with constants

$$\kappa = 0.41, \quad \tilde{\zeta}_2 = 1.755, \quad \sigma_\phi = \frac{2}{3}, \quad F_{\mathrm{SAS}} = 1.25.$$

4.9 Implementation details

4.9.1 Limitation of *k* and *omega*

For an increased robustness a limitation on the turbulent quantities is employed, which is designed to let *k* and *omega* unchanged for a converged solution but to limit the quantities during the transitional phase or to avoid not physical values (as e.g. negative values), which may occur locally. A maximum and a minimum threshold on the turbulent intensity is used to bound the turbulent kinetic energy. The upper limit is defined such it should not be reached. The cut of at the lower limit should not influence the solution:

$$\begin{aligned} TU_{max} &= 0.800, \\ TU_{min} &= 1.e-8 \end{aligned} \tag{4.70}$$

This implies absolute limits for the *k* in relation to the local velocity field:

$$\begin{aligned} k_{max} &= 1.5 * TU_{max}{}^2 * (u^2+v^2+w^2+u_{eps}{}^2) \\ k_{min} &= 1.5 * TU_{min}{}^2 * (u^2+v^2+w^2) \end{aligned} \tag{4.71}$$

with $u_{eps}{}^2 = 0.05 * u_{ref}{}^2$, which is usually a negligible contribution to this term, but it avoids k_{max} going to small in flow regions with small velocities. The local *k* is used after its limitation to find a lower bound for ω

$$\omega_{min} = MAX(\sqrt{\frac{k}{d^2}}, \frac{\rho k}{\mu_{t,max}}) \tag{4.72}$$

with $\mu_{t,max}$ being an upper limit for the eddy viscosity (parameter input), which should not be set lower than some thousand times the laminar viscosity (default is $20000 \cdot \mu_t$). $\bar{d}$ is the local wall distance for $\bar{d} > \delta$ and is limited to $\bar{d} = \delta$ elsewhere in order not to limit ω too strong very near the wall inside the boundary layer. δ is a characteristic length related to the boundary layer thickness.

4.9.2 Initial, freestream and boundary conditions

As initial conditions, when starting from scratch farfield values are assigned to all grid points. Farfield (or inflow) conditions for *k* and *omega* are

$$\begin{aligned} k_\infty &= 1.5 * TU_\infty{}^2 * u_\infty{}^2 \\ \omega_\infty &= \frac{k_\infty}{\mu_{t,\infty}} \end{aligned} \tag{4.73}$$

In case of an inflow boundary (e.g. engine exhaust condition) the related reference values of the specific boundary are used instead of the infinity-values. The wall boundary conditions are

$$\begin{aligned} k_{wall} &= 0.0 \\ \omega_{wall} &= 10 * (\frac{6\mu_l}{D^2\beta_{k1}}) \end{aligned} \tag{4.74}$$

with the model constant $\beta_{k1} = 0.09$ and D being the distance of the near point to the wall (where y^+ is measured), which should satisfy the requirement to have a smooth wall sublayer behavior at least if y^+ is lower than 3.

4.10 Transition modeling

In order to guarantee laminar flow over surfaces which are flagged for laminar treatment, we limit the production in those regions to be smaller or equal the turbulent destruction. Thus, the information is needed for each node, if it is closer to a 'laminar' or to a 'turbulent' no slip wall. We simply enhance the property of the variable d (see equation 4.25) by computing $\tilde{d}$:
$\tilde{d} = -d$ for field points near a 'laminar no slip wall'
$\tilde{d} = +d$ for field points near a 'turbulent no slip wall'
Now, the wall distance is $d = |\tilde{d}|$
In addition we introduced the parameter δ_{max}. On points with $\tilde{d} < 0$ and $|\tilde{d}| < \delta_{max}$ we perform the above described limitation. The default value of δ_{max} is infinity. Using this value the parameter has no effect. However, it can be used to bound the region of limiting the production of turbulence.

4.11 Wall functions (Near-wall modeling)

The aim of hybrid (or grid-independent) wall-functions is to provide a boundary condition at solid walls that enables flow solutions independent of the location of the first grid node above the wall. The two classical wall boundary conditions are low-Re and high-Re type boundary conditions. The *low-Re boundary condition* imposes no-slip at the wall and requires a low-Re grid with $y^+(1) \approx 1$. The *hybrid-Re boundary condition* is an improved *high-Re* boundary condition, prescribing the wall-shear stress and no-penetration at the wall. The RANS equations are solved only down to the first grid node above the wall and are matched with an adaptive wall function solution at the first grid node above the wall. The low-Re restriction is no longer valid for hybrid-Re b.c.
The implementation can be most easily understood by starting with

$$\frac{d}{dt}\int_V \vec{W}\, dV \;=\; -\int_{\partial V} \mathbb{F}\cdot\vec{n}\, dS \;+\; \int_{\partial V} \mathbb{F}_v\cdot\vec{n} dS$$

where V is an arbitrary control volume with closed boundary surface ∂V, and $\vec{n}$ is the unit normal vector in outward direction. The vector of conservative variables $\vec{W}$, and the convective and viscous flux tensor $\mathbb{F}$ and $\mathbb{F}_v$ resp. are given by

$$\vec{W} \;=\; \begin{pmatrix} \rho \\ \rho\vec{u} \\ \rho(c_v\theta + \frac{1}{2}\vec{u}\cdot\vec{u}) \end{pmatrix}, \quad \mathbb{F} \;=\; \begin{pmatrix} \rho\vec{u} \\ \rho\vec{u}\otimes\vec{u} \;+\; p\mathbb{I} \\ \rho E\vec{u} + p\vec{u} \end{pmatrix}, \quad \mathbb{F}_v \;=\; \begin{pmatrix} 0 \\ \mathbb{T} \\ \mathbb{T}\,\vec{u} - \vec{q} \end{pmatrix}$$

If V is a boundary cell then the inviscid fluxes across the boundary vanish and the task is to impose suitable viscous fluxes. The boundary cells are half cells in the sense that the corresponding grid nodes are located on the boundary. For the standard low-Re boundary condition, the fluxes across the wall are set to zero and the correct Dirichlet values are prescribed for the conservative variables momentum and energy (if the wall is isothermal).
Concerning the hybrid wall-function boundary condition, the standard approach for cell-centered

type FVM, i.e., to simply modify μ_t in the boundary cell, cannot be used. Therefore, as a boundary condition the fluxes across the wall are prescribed. We use the approximation

$$\mathbb{T}\cdot\vec{n} \;=\; (\mathbb{I}-\vec{n}\otimes\vec{n})\,\mathbb{T}\;\cdot\;\vec{n}\;+\;\vec{n}\otimes\vec{n}\,\mathbb{T}\;\cdot\;\vec{n}\;\approx\;-\tau_w\vec{u}_t,$$

where $\vec{u}_t$ is a vector whose direction is given by the projection of the velocity (at the first node above the wall) parallel to the wall and which is of unit magnitude. The normal contribution of the wall-shear stress is neglected. Now we describe how τ_w is computed.
We introduce the non-dimensional variables (so-called plus units)

$$u^+=\frac{u}{u_\tau}\,,\quad y^+=\frac{yu_\tau}{\nu}\,,\quad \nu_t^+=\frac{\nu_t}{\nu}\,,\quad \tilde{\nu}^+=\frac{\tilde{\nu}}{\nu}\,,\quad k^+=\frac{k}{u_\tau^2}\,,\quad \omega^+=\frac{\omega\nu}{u_\tau^2}$$

Suppose a universal hybrid law of the wall for velocity (i.e., magnitude of wall parallel velocity as a function of distance from the wall) is known in either of the two forms

$$u^+=F(y^+)\;\Leftrightarrow\;\frac{u}{u_\tau}=F\left(\frac{yu_\tau}{\nu}\right)\;\text{ resp. }\;y^+=F^{-1}(u^+)\;\Leftrightarrow\;\frac{yu_\tau}{\nu}=F^{-1}\left(\frac{u}{u_\tau}\right).$$

Then the matching condition at each first node above the wall (at wall-distance y_δ, say) dictates that the wall-parallel components of the RANS solution u_{RANS} and the wall-function u_{WF} are equal at wall distance y_δ

$$u_{RANS}(y_\delta)\;=\;u_{WF}(y_\delta)$$

From the universal wall law $u_{WF}(y)=u_\tau F(yu_\tau/\nu)$, so we obtain

$$F\left(\frac{y_\delta u_\tau}{\nu}\right)=\frac{u_{RANS}(y_\delta)}{u_\tau}\quad\text{resp.}\quad F^{-1}\left(\frac{u_{RANS}(y_\delta)}{u_\tau}\right)=\frac{y_\delta u_\tau}{\nu}$$

which can be solved for u_τ using Newton's method and then we set $\tau_w=\rho u_\tau^2$.
We use one wall law for all SA-models and one wall-law for all k-ω models, viz.,

$$F_{SA,a}=(1-\phi_{SA})F_{Sp,5}+\phi_{SA}F_{Rei,m}\,,\;\;\phi_{SA}=\tanh(arg)\,,\;\;arg=\left(\frac{y^+}{24}\right)^3$$

$$F_{k\omega,a}=(1-\phi_{k\omega})F_{Sp,3}+\phi_{k\omega}F_{Rei,m}\,,\;\;\phi_{k\omega}=\tanh(arg)\,,\;\;arg=\left(\frac{y^+}{50}\right)^2$$

which are constructed via blending from the following classical wall laws

$$F_{Rei,m}=(1-\phi_{b1})F_{Rei}+\phi_{b1}F_{log}\,,\;\;\phi_{b1}=\tanh(arg)\,,\;\;arg=\left(\frac{y^+}{27}\right)^4$$

$$y^+=F_{Sp,N}^{-1}(u^+)\;\text{ with }\;F_{Sp,N}^{-1}(u^+)\equiv u^+ + e^{-\kappa B}\left(e^{\kappa u^+}-\sum_{n=0}^{N}\frac{(\kappa u^+)^n}{n!}\right)$$

$$u^+=F_{Rei}(y^+)\;\text{ with}$$

$$F_{Rei}(y^+)\equiv\frac{\ln(1+0.4y^+)}{\kappa}+7.8\left(1-e^{-\frac{y^+}{11.0}}-\frac{y^+}{11.0}e^{-\frac{y^+}{3.0}}\right)$$

$$F_{Rei,m}=(1-\phi_{b1})F_{Rei}+\phi_{b1}F_{log}\,,\;\;\phi_{b1}=\tanh(arg)\,,\;\;arg=\left(\frac{y^+}{27}\right)^4$$

The turbulent heat flux $\dot{q}_w$ is then computed from the relation

$$\dot{q}_w\;=\;\frac{c_p u_\tau(\theta_W-\theta(y_\delta))}{\theta^+(y_\delta^+)},$$

with wall temperature θ_W, temperature at first node above the wall $\theta(y_\delta)$ and wall-function $\theta^+(y_\delta^+)$ given by

$$\theta^+(y^+) \equiv \begin{cases} Pr\, y^+, & \text{if } y^+ \le R_\theta \\ C_\theta(2\ln(\frac{y^+}{R_\theta})+1)^{\frac{1}{2}}, & \text{if } y^+ > R_\theta \end{cases},$$

with $R_\theta = 8.0$, $C_\theta = PrR_\theta$, which is by smooth at $y^+ = R_\theta$ by construction.
The boundary conditions for $\tilde{\nu}$ in the SA type models and for k in the k-ω-type models remain unchanged. As far as the boundary condition for ω is concerned, it turns out to be superior to use the approach by Wilcox: We impose ω at the first node above the wall (and correspondingly set the fluxes for ω to zero for the first cell above the wall)

$$\text{At first off-wall node:} \quad \omega = \phi\omega_{b1} + (1-\phi)\omega_{b2}\ ,\ \ \phi = \tanh(arg^4)\ ,\ \ arg = \frac{y^+}{10}$$

with the blending formula

$$\omega_{b1} = \omega_{vis} + \omega_{log}\ , \quad \omega_{b2} = \left(\omega_{vis}^{1.2} + \omega_{log}^{1.2}\right)^{1/1.2}$$

and the asymptotic relations

$$\text{Asymptotic:} \quad \omega_{vis} = \frac{6\nu}{\beta y^2}\ , \quad \omega_{log} = \frac{u_\tau}{\sqrt{\beta_k}\kappa y}.$$

5 Accuracy Improvements

5.1 Introduction

The performance of many existing compressible codes, originally implemented for transonic or supersonic problems, degrades as the Mach number of the computed flow tends to zero. Moreover, the lower the Mach number is, the more the accuracy of the solution and the more the iterative convergence degrades. However, the need of computing low Mach number flows or locally compressible flows is more and more frequently encountered in engineering. This is among other things due to the fact that when the variations of the temperature cannot be neglected the incompressible flow equations can no longer be used even if the onflow Mach number is low.

A typical example for such a configuration in aerodynamics is the flow over a multi-element airfoil at high angle of attack. In Figure 5.1 it can be seen that although the flow against the airfoil is of low Mach number and a spacious part of the flow region is incompressible with locally low Mach numbers, there occur significant compressibility effects in other regions of the flow, in particular in the neighborhood of the airfoil.

To describe such flow effects in this chapter we consider for an open domain $D \subset \mathbb{R}^3$ the Euler equations for a three-dimensional flow

$$\mathbf{u}(\mathbf{x},t) = (u_1(\mathbf{x},t), u_2(\mathbf{x},t), u_3(\mathbf{x},t)), \qquad (\mathbf{x},t) \in D \times (0,\infty)$$

in conservative variables $\mathbf{w}_1 := (\rho, \rho u_1, \rho u_2, \rho u_3, \rho E)$ written as

$$\frac{\partial \mathbf{w}_1}{\partial t} + \sum_{i=1}^{3} \frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = 0, \qquad \mathbf{x} = (x_1, x_2, x_3), \tag{5.1}$$

where the mappings $\mathbf{f}^{(i)} := \mathbf{f}^{(i)}(\mathbf{w}_1)$, $i = 1,2,3$, are given by

$$\mathbf{f}^{(1)} = \begin{pmatrix} \rho u_1 \\ \rho u_1^2 + p \\ \rho u_1 u_2 \\ \rho u_1 u_3 \\ \rho H u_1 \end{pmatrix}, \quad \mathbf{f}^{(2)} = \begin{pmatrix} \rho u_2 \\ \rho u_1 u_2 \\ \rho u_2^2 + p \\ \rho u_2 u_3 \\ \rho H u_2 \end{pmatrix} \quad \text{and} \quad \mathbf{f}^{(3)} = \begin{pmatrix} \rho u_3 \\ \rho u_1 u_3 \\ \rho u_2 u_3 \\ \rho u_3^2 + p \\ \rho H u_3 \end{pmatrix}. \tag{5.2}$$

Here the quantities $\rho := \rho(\mathbf{x},t)$, $E := E(\mathbf{x},t)$ and $H = E + p/\rho$ describe the density, the total Energy per unit mass and the total enthalpy per unit mass. The pressure $p := p(\mathbf{x},t)$ is given by the equation of state for a perfect gas

$$p = (\gamma - 1)\rho \left(E - \frac{\|\mathbf{u}\|_2^2}{2} \right) \tag{5.3}$$

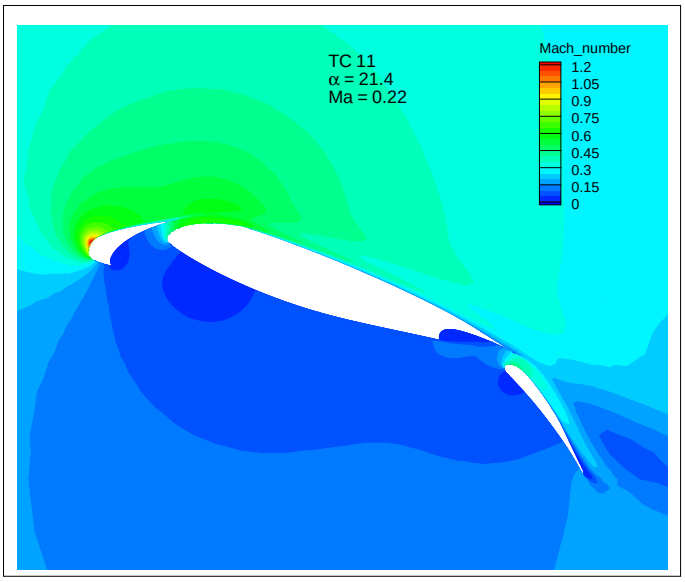


Figure 5.1: Flow over a multi-element airfoil at angle of attack $\alpha = 21.4°$ and inflow Mach number 0.22

where γ denotes the ratio of specific heats and $\|\mathbf{u}\|_2$ the Euclidean norm of the fluid speed,

$$\|\mathbf{u}\|_2 = \sqrt{u_1^2 + u_2^2 + u_3^2}.$$

Our goal is to find an approximate solution of (5.1) for flow problems of mixed compressible and incompressible type that is part of the flow region is incompressible with locally low Mach numbers, whereas there occur significant compressibility effects in other regions of the flow. For simplicity in our analysis we only consider Euler's equation. The examples presented in Section 5.7 also include Navier-Stokes test cases.

Physically the difficulty in solving the compressible Euler equations (5.1) for low Mach numbers are associated with the large disparity of the acoustic wave speed and the waves convected at fluid speed. Mathematically the resulting system of equations is stiff and the allowed time step size to compute an approximate steady state solution of (5.1) is usually so small that convergence of the iterates is too slow. Moreover, usually the accuracy of computed values such as the C-lift or C-drag is deteriorated. The stiffness of the system can either be shown by a spectral analysis of Euler's equation (see Section 5.3) or by a perturbation analysis of the low Mach number limit (see [73, Chapter 2]).

To this end several so called preconditioning methods have been suggested by many authors to improve the convergence properties of explicit Runge-Kutta methods applied to solve (5.1) (see e.g. [70, 78, 16, 5]). All these methods are based on the following idea. We are interested in a steady state solution of (5.1) that is a solution which satisfies

$$\sum_{i=1}^{3} \frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = 0.$$

Hence, we can multiply the time derivative of (5.1) with some matrix valued mapping $P := P(\mathbf{w}_1) \in \mathbb{R}^{5\times 5}$ which approximately slows the speed of the acoustic waves down towards the fluid speed. In the literature these mappings are called preconditioners. They often do not depend only on the local values of the fluid, furthermore several parameters need to be determined to ensure efficiency of these preconditioners when applied in numerical algorithms for finding an approximate solution of (5.1). Mathematically these preconditioners try to reduce the

stiffness of the original system to improve the convergence speed of the explicit Runge-Kutta methods applied to compute a steady state solution of (5.1).

Although many articles have been concerned with the definition of preconditioners to the author's experience only a few work has been spent on a mathematical investigation on the effect these preconditioners have in a numerical scheme. Furthermore, often the choice of appropriate parameters seems to be an open problem.

Besides explicit Runge-Kutta methods one considers implicit methods to solve (5.1). These methods have the advantage that the time step size to compute an approximate solution can be chosen much larger when compared with an explicit scheme. Hence, we expect that for an implicit scheme no preconditioning is necessary to reach desirable convergence rates. On the other hand, general implicit methods require a high amount of fast memory and usually a nonlinear system of equations needs to be solved in each time step. For large scale three-dimensional flow problems the complexity and time period to setup and solve such systems is often not acceptable or it is even impossible. Moreover, frequently a good initial guess for Newton's method applied to solve the nonlinear system is not at hand. Therefore, during the last few years methods which lie in between explicit and implicit Runge-Kutta methods have been developed to overcome these problems. One of these methods, the so-called LUSGS-method (see Dwight [20] for more details and Figures 5.11–5.13), is implemented in the DLR TAU-Code. For an investigation of implicit upwind schemes we refer to [74]. Unfortunately such kind of methods frequently do not have the same stability when compared with the explicit methods.

In this chapter we consider preconditioning methods for explicit Runge-Kutta methods applied to find approximate steady state solutions of (5.1). In Section 5.2 we recall some theoretical results concerning Euler's equation. Moreover, since we want to have some freedom in the definition of certain preconditioners we reformulate Euler's equation with respect to an appropriate set of variables. Section 5.3 deals with a spectral analysis of Euler's equation and the definition of a preconditioner. This preconditioner can be interpreted as a generalization on many other preconditioners discussed in the literature so far. The effect of the preconditioner on the spectrum will be investigated in Section 5.4. The choice of the parameters occurring in the preconditioners is the topic of Section 5.5. Moreover, in the latter section we will also suggest some parameter choice rules for the preconditioner. In Section 5.7 we will compare in numerical examples the different parameter choice rules. In particular, we will show that a new implemented parameter choice rule yields numerically more stability than the one which was originally implemented in the DLR TAU-Code.

5.2 Representation of preconditioners

Before we discuss preconditioners for the Euler equation (5.1) let us give some comments concerning their implementation. Assume we have a formulation of Euler's equation (5.1) in some set of variables $\mathbf{q}$, that is (5.1) is replaced by

$$\frac{\partial \mathbf{q}}{\partial t} + \sum_{i=1}^{3} A_{\mathbf{q}}^{(i)} \frac{\partial \mathbf{q}}{\partial x_i} = 0, \tag{5.4}$$

where

$$A_{\mathbf{q}}^{(i)} = \frac{\partial \mathbf{f}^{(i)}}{\partial \mathbf{q}}, \qquad i = 1,2,3. \tag{5.5}$$

On the other hand, for example due to a theoretical analysis or only for simplicity, a preconditioner $P(\mathbf{w})$ is formulated for another set of variables $\mathbf{w}$. Hence, to apply $P(\mathbf{w})$ we need to carry equation (5.4) over into the variables $\mathbf{w}$ and multiplicate the second term on the left hand side by the preconditioner, that is we replace (5.4) by

$$\frac{\partial \mathbf{w}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial t} + P(\mathbf{w})\sum_{i=1}^{3} A_{\mathbf{w}}^{(i)}\frac{\partial \mathbf{w}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial x_i} = 0.$$

Since $(\partial \mathbf{w}/\partial \mathbf{q})^{-1} = \partial \mathbf{q}/\partial \mathbf{w}$ we obtain

$$\frac{\partial \mathbf{q}}{\partial t} + P(\mathbf{q})\sum_{i=1}^{3} A_{\mathbf{q}}^{(i)}\frac{\partial \mathbf{q}}{\partial x_i} = 0, \tag{5.6}$$

where

$$P(\mathbf{q}) := \frac{\partial \mathbf{q}}{\partial \mathbf{w}}P(\mathbf{w})\frac{\partial \mathbf{w}}{\partial \mathbf{q}} \qquad \text{and} \qquad A_{\mathbf{q}}^{(i)} := \frac{\partial \mathbf{q}}{\partial \mathbf{w}}A_{\mathbf{w}}^{(i)}\frac{\partial \mathbf{w}}{\partial \mathbf{q}} \tag{5.7}$$

denote the operators with respect to the change of coordinates. In particular, usually Euler's equation is given in conservative variables $\mathbf{q} = \mathbf{w}_1$, which yields using (5.5)

$$\frac{\partial \mathbf{w}_1}{\partial t} + P(\mathbf{w}_1)\sum_{i=1}^{3}\frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = 0. \tag{5.8}$$

Now, if we want to carry (5.8) over into another set of variables $\mathbf{r}$ and the preconditioner is only given for the set of variables $\mathbf{w}$, as above using the formula (5.7) for $P(\mathbf{w})$ we obtain from (5.8)

$$\frac{\partial \mathbf{r}}{\partial t} + \frac{\partial \mathbf{r}}{\partial \mathbf{w}_1}\frac{\partial \mathbf{w}_1}{\partial \mathbf{w}}P(\mathbf{w})\frac{\partial \mathbf{w}}{\partial \mathbf{w}_1}\sum_{i=1}^{3}\frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = \frac{\partial \mathbf{r}}{\partial t} + \Gamma(\mathbf{r},\mathbf{w})\sum_{i=1}^{3}\frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = 0$$

where

$$\Gamma(\mathbf{r},\mathbf{w}) := \frac{\partial \mathbf{r}}{\partial \mathbf{w}}P(\mathbf{w})\frac{\partial \mathbf{w}}{\partial \mathbf{w}_1}. \tag{5.9}$$

Altogether, if we have formulated Euler's equation (5.1) in conservative variables $\mathbf{w}_1$ and if we have a preconditioner for the set of variables $\mathbf{w}$ and if we want to have a formulation of Euler's equation in the set of variables $\mathbf{r}$, the corresponding Euler's equation is given by (5.9) and the preconditioner is denoted by $\Gamma(\mathbf{r},\mathbf{w})$. Finally, we need to rewrite the time derivative into the conservative variables $\mathbf{w}_1$, that is we solve the preconditioned Euler equation

$$\frac{\partial \mathbf{w}_1}{\partial t} + \frac{\partial \mathbf{w}_1}{\partial \mathbf{r}}\Gamma(\mathbf{r},\mathbf{w})\sum_{i=1}^{3}\frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = 0. \tag{5.10}$$

These considerations give us some freedom in the formulation and for the analysis of preconditioners since it allows the choice of any suitable variables.

To solve Euler's equation (5.1) we apply a central difference scheme. To ensure numerical stability additional artificial dissipative terms are required. One usually adds a second difference to control oscillations near shocks and a fourth difference to damp high frequency oscillations (see Jameson et al. [32]). That is, instead of (5.10) we may solve the equation

$$\frac{\partial \mathbf{w}_1}{\partial t} + \frac{\partial \mathbf{w}_1}{\partial \mathbf{r}}\Gamma(\mathbf{r},\mathbf{w})\sum_{i=1}^{3}\frac{\partial}{\partial x_i}\left(\mathbf{f}^{(i)} + \Gamma(\mathbf{r},\mathbf{w})^{-1}N_D^{(i)}(\mathbf{r};\kappa_2,\kappa_4)\right) = 0 \tag{5.11}$$

where

$$N_D^{(i)}(\mathbf{r};\kappa_2,\kappa_4) := \kappa_2\rho(P(\mathbf{r})A_{\mathbf{r}}^{(i)})\frac{\partial \mathbf{r}}{\partial x_i} + \kappa_4\rho(P(\mathbf{r})A_{\mathbf{r}}^{(i)})\frac{\partial^3 \mathbf{r}}{\partial x_i^3}$$

and

$$\rho(A) := \max\{|\lambda| : \det(\lambda I - A) = 0\}$$

denotes the spectral radius of a matrix $A \in \mathbb{R}^{n\times n}$. The values $\kappa_2, \kappa_4 \geq 0$ are usually fixed parameters which are chosen beforehand. In the literature many other possibilities to determine an artificial dissipation for numerical stability have been suggested. Note that at the outset any reasonable method approximating the integrals with taking care of the flow field

$$\int_{\partial\Omega} \mathbf{f}^{(i)}(\mathbf{w}_1) n_i ds(y)$$

can be used to stabilize a numerical scheme to solve (5.1).

5.3 Spectral Analysis of Euler's equation

Let us start this section by carrying out a spectral analysis of Euler's equation. For this purpose an appropriate set of variables is either given by the so-called entropy variables

$$\mathbf{w}_0 := (p, u_1, u_2, u_3, s),$$

whereby s denotes the entropy determined as $s = \ln(p/\rho^\gamma)$ or by the primitive variables

$$\mathbf{w}_4 := (p, u_1, u_2, u_3, T),$$

where T denotes the temperature. We use the latter set of variables. Then, by a change of coordinates (5.1) reads in the variables $\mathbf{w}_4$

$$\frac{\partial \mathbf{w}_4}{\partial t} + \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}\sum_{i=1}^{3}\frac{\partial \mathbf{f}^{(i)}}{\partial \mathbf{w}_4}\frac{\partial \mathbf{w}_4}{\partial x_i} = 0. \tag{5.12}$$

To obtain explicit expressions for the mappings

$$B_4^{(i)} := \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}\frac{\partial \mathbf{f}^{(i)}}{\partial \mathbf{w}_4}, \qquad i = 1,2,3, \tag{5.13}$$

an explicit calculation of the terms $\partial \mathbf{f}^{(i)}/\partial \mathbf{w}_4$ and $\partial \mathbf{w}_4/\partial \mathbf{w}_1$ is required. To explicitly compute $\partial \mathbf{w}_1/\partial \mathbf{w}_4$ and its inverse $\partial \mathbf{w}_4/\partial \mathbf{w}_1$ we use the definitions

$$z_1 := \rho, \quad z_2 := \rho u_1, \quad z_3 := \rho u_2, \quad z_4 := \rho u_3 \quad \text{and} \quad z_5 := \rho E \tag{5.14}$$

and consider the mapping

$$\xi : \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{pmatrix} \mapsto \begin{pmatrix} (\gamma-1)\left(z_5 - \frac{z_2^2+z_3^2+z_4^2}{2z_1}\right) \\ \frac{z_2}{z_1} \\ \frac{z_3}{z_1} \\ \frac{z_4}{z_1} \\ (\gamma-1)\left(\frac{z_5}{z_1} - \frac{z_2^2+z_3^2+z_4^2}{2z_1^2}\right) \end{pmatrix}.$$

A straightforward computation shows that ξ maps the conservative variables $\mathbf{w}_1$ to the primitive variables $\mathbf{w}_4$, that is $\xi(\mathbf{w}_1) = \mathbf{w}_4$. For instance, due to the equation of state for a perfect gas (5.3) we have

$$\begin{aligned} \rho E &= \frac{p}{\gamma-1} + \frac{\|\mathbf{u}\|_2^2}{2} \\ &= \frac{p}{\gamma-1} + \frac{1}{2}\left(\frac{z_2^2}{z_1} + \frac{z_3^2}{z_1} + \frac{z_4^2}{z_1}\right). \end{aligned}$$

and therefore

$$p = (\gamma-1)\left[z_5 - \frac{1}{2}\left(\frac{z_2^2}{z_1} + \frac{z_3^2}{z_1} + \frac{z_4^2}{z_1}\right)\right]. \tag{5.15}$$

Now, using the relation $\partial\mathbf{w}_4/\partial\mathbf{w}_1 = \partial\xi/\partial\mathbf{z}$ we have

$$\begin{aligned} \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1} &= \begin{pmatrix} (\gamma-1)\frac{z_2^2+z_3^2+z_4^2}{2z_1^2} & (1-\gamma)\frac{z_2}{z_1} & (1-\gamma)\frac{z_3}{z_1} & (1-\gamma)\frac{z_4}{z_1} & (\gamma-1) \\ -\frac{z_2}{z_1^2} & \frac{1}{z_1} & 0 & 0 & 0 \\ -\frac{z_3}{z_1^2} & 0 & \frac{1}{z_1} & 0 & 0 \\ -\frac{z_4}{z_1^2} & 0 & 0 & \frac{1}{z_1} & 0 \\ (\gamma-1)\left(\frac{z_2^2+z_3^2+z_4^2}{z_1^3} - \frac{z_5}{z_1^2}\right) & (1-\gamma)\frac{z_2}{z_1^2} & (1-\gamma)\frac{z_3}{z_1^2} & (1-\gamma)\frac{z_4}{z_1^2} & \frac{\gamma-1}{z_1} \end{pmatrix} \\ &= \begin{pmatrix} (\gamma-1)\frac{\|\mathbf{u}\|_2^2}{2} & (1-\gamma)u_1 & (1-\gamma)u_2 & (1-\gamma)u_3 & (\gamma-1) \\ -\frac{u_1}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{u_2}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{u_3}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ \frac{1}{\rho}\left((\gamma-1)\frac{\|\mathbf{u}\|_2^2}{2} - T\right) & (1-\gamma)\frac{u_1}{\rho} & (1-\gamma)\frac{u_2}{\rho} & (1-\gamma)\frac{u_3}{\rho} & \frac{\gamma-1}{\rho} \end{pmatrix}. \end{aligned}$$

The entry $(\partial\mathbf{w}_4/\partial\mathbf{w}_1)_{5,1}$ of this matrix follows from the definition of E and

$$\begin{aligned} (\gamma-1)\left(\frac{z_2^2+z_3^2+z_4^2}{z_1^3} - \frac{z_5}{z_1^2}\right) &= \frac{\gamma-1}{\rho}\left(\frac{\|u\|_2^2}{2} - \frac{p}{(\gamma-1)\rho}\right) \\ &= \frac{1}{\rho}\left((\gamma-1)\frac{\|u\|_2^2}{2} - T\right). \end{aligned}$$

The computation of the other entries is trivial. To determine an explicit expression for $\partial\mathbf{f}^{(1)}/\partial\mathbf{w}_4$ we reformulate $\mathbf{f}^{(1)}$ in terms of the variables $\mathbf{w}_4$, that is

$$\mathbf{f}^{(1)} = \begin{pmatrix} \frac{p}{T}u_1 \\ \frac{p}{T}u_1^2 + p \\ \frac{p}{T}u_1u_2 \\ \frac{p}{T}u_1u_3 \\ \left[\frac{p}{T}\left(\frac{T}{\gamma-1} + \frac{u_1^2+u_2^2+u_3^2}{2}\right) + p\right]u_1 \end{pmatrix}$$

and obtain

$$\frac{\partial \mathbf{f}^{(1)}}{\partial \mathbf{w}_4} = \begin{pmatrix} \frac{u_1}{T} & \frac{p}{T} & 0 & 0 & -\frac{pu_1}{T^2} \\ \frac{u_1^2}{T} + 1 & 2u_1\frac{p}{T} & 0 & 0 & -\frac{pu_1^2}{T^2} \\ \frac{u_1u_2}{T} & \frac{p}{T}u_2 & \frac{p}{T}u_1 & 0 & -\frac{pu_1u_2}{T^2} \\ \frac{u_1u_3}{T} & \frac{p}{T}u_3 & 0 & \frac{p}{T}u_3 & -\frac{pu_1u_3}{T^2} \\ \left[\frac{1}{T}\left(\frac{T}{\gamma-1} + \frac{\|\mathbf{u}\|_2^2}{2}\right) + 1\right]u_1 & \frac{p}{T}\left(\frac{T}{\gamma-1} + \frac{\|\mathbf{u}\|_2^2}{2} + p + u_1^2\right) & \frac{p}{T}u_1u_2 & \frac{p}{T}u_1u_3 & \frac{-p\|\mathbf{u}\|_2^2}{2T^2}u_1 \end{pmatrix}.$$

To illustrate some of the computations we explicitly compute some of the elements of the matrix above, for instance

$$\begin{aligned}
\left(\frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}\frac{\partial \mathbf{f}_1^1}{\partial \mathbf{w}_4}\right)_{1,1} &= (\gamma-1)\frac{\|\mathbf{u}\|_2^2 u_1}{2T} + (1-\gamma)u_1\left(\frac{u_1^2}{T}+1\right) + (1-\gamma)\frac{u_1u_2^2}{T} \\
&\quad +(1-\gamma)\frac{u_1u_3^2}{T} + (\gamma-1)\left[\frac{1}{T}\left(\frac{T}{\gamma-1}+\frac{\|\mathbf{u}\|_2^2}{2}\right)+1\right]u_1 \\
&= (\gamma-1)\frac{\|\mathbf{u}\|_2^2 u_1}{T} + (1-\gamma)\left[\frac{\|\mathbf{u}\|_2^2 u_1}{T}+u_1\right] + (\gamma-1)u_1 + u_1 \\
&= u_1, \\
\left(\frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}\frac{\partial \mathbf{f}_1^1}{\partial \mathbf{w}_4}\right)_{1,2} &= (\gamma-1)\frac{\|\mathbf{u}\|_2^2}{2}\frac{p}{T} + 2(1-\gamma)u_1^2\frac{p}{T} + (1-\gamma)u_2^2\frac{p}{T} \\
&\quad +(1-\gamma)u_3^2\frac{p}{T} + (\gamma-1)\frac{p}{T}\left(\frac{T}{\gamma-1}+\frac{\|\mathbf{u}\|_2^2}{2}+p+u_1^2\right), \\
\left(\frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}\frac{\partial \mathbf{f}_1^1}{\partial \mathbf{w}_4}\right)_{1,5} &= (1-\gamma)\frac{\|\mathbf{u}\|_2^2}{2}\frac{pu_1}{T^2} + (\gamma-1)u_1\frac{pu_1^2}{T^2} + (\gamma-1)u_2\frac{pu_1u_2}{T^2} \\
&\quad +(\gamma-1)u_3\frac{pu_1u_3}{T^2} + (1-\gamma)\frac{\|\mathbf{u}\|_2^2}{2}\frac{pu_1}{T^2} \\
&= 0
\end{aligned}$$

Finally, we obtain

$$B_4^{(1)} = \begin{pmatrix} u_1 & \rho a^2 & 0 & 0 & 0 \\ \frac{1}{\rho} & u_1 & 0 & 0 & 0 \\ 0 & 0 & u_1 & 0 & 0 \\ 0 & 0 & 0 & u_1 & 0 \\ 0 & (\gamma-1)T & 0 & 0 & u_1 \end{pmatrix}$$

and by symmetry arguments

$$B_4^{(2)} = \begin{pmatrix} u_2 & 0 & \rho a^2 & 0 & 0 \\ 0 & u_2 & 0 & 0 & 0 \\ \frac{1}{\rho} & 0 & u_2 & 0 & 0 \\ 0 & 0 & 0 & u_2 & 0 \\ 0 & 0 & (\gamma-1)T & 0 & u_2 \end{pmatrix}, \quad B_4^{(3)} = \begin{pmatrix} u_3 & 0 & 0 & \rho a^2 & 0 \\ 0 & u_3 & 0 & 0 & 0 \\ 0 & 0 & u_3 & 0 & 0 \\ \frac{1}{\rho} & 0 & 0 & u_3 & 0 \\ 0 & 0 & 0 & (\gamma-1)T & u_3 \end{pmatrix}$$

where a denotes the speed of sound defined through $a^2 := \frac{\gamma p}{\rho}$.

It follows by straightforward computations that the eigenvalues of $B_4^{(1)}$ are given by the diagonal elements of the matrix $D_4^{(1)} := \mathrm{diag}(u_1, u_1, u_1, u_1+a, u_1-a)$ and the corresponding eigenvectors are the columns of

$$V_4^{(1)} := \begin{pmatrix} 0 & 0 & 0 & \frac{\rho a^2}{(\gamma-1)T} & \frac{\rho a^2}{(\gamma-1)T} \\ 0 & 0 & 0 & \frac{a}{(\gamma-1)T} & -\frac{a}{(\gamma-1)T} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Moreover, again by symmetry arguments we conclude that the eigenvalues of $B_4^{(2)}$ and $B_4^{(3)}$ are given by the diagonal elements of $D_4^{(2)} := \mathrm{diag}(u_2, u_2, u_2, u_2+a, u_2-a)$, and $D_4^{(3)} := \mathrm{diag}(u_3, u_3, u_3, u_3+$

$a, u_3 - a$) and the corresponding eigenvectors are the columns of the matrices

$$V_4^{(2)} := \begin{pmatrix} 0 & 0 & 0 & \frac{\rho a^2}{(\gamma-1)T} & \frac{\rho a^2}{(\gamma-1)T} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{a}{(\gamma-1)T} & -\frac{a}{(\gamma-1)T} \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}, \quad V_4^{(3)} := \begin{pmatrix} 0 & 0 & 0 & \frac{\rho a^2}{(\gamma-1)T} & \frac{\rho a^2}{(\gamma-1)T} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{a}{(\gamma-1)T} & -\frac{a}{(\gamma-1)T} \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

From this analysis it can be seen that the difficulty in solving the compressible equations for low Mach numbers is associated with the large disparity of the acoustic wave speed, $u_i + a$ and the waves convected at the fluid speed, u_i. Consequently, the resulting system is stiff and cannot be solved straightforward by explicit Runge-Kutta methods in general. Since we are only interested in steady state solutions of (5.1) the basic idea to overcome this problem is to premultiply the time derivative by a preconditioner such that the acoustic and convected wave speeds are locally clustered. Mathematically this reduces the stiffness of the equations and allows larger time step sizes.

5.4 Definition of preconditioners

In the literature several preconditioners have been suggested. Within this context these are mappings $P(\mathbf{w}_4) := P(\alpha, \beta, \delta, \mathbf{w}_4) \in \mathbb{R}^{5\times 5}$ which are regular for all choices of parameters $(\alpha, \beta, \delta) \in \Omega \subset \mathbb{R}^3$ and $\mathbf{w}_4$. Here Ω denotes some set from which the parameters can be chosen. Naturally, we can represent P by formula (5.7) for any other set of variables.

In [57] the following preconditioner can be found which is a generalization of many other examined preconditioners,

$$P(\mathbf{w}_4) := \begin{pmatrix} m^2 g & 0 & 0 & 0 & -m^2 \frac{\gamma p}{T}\delta \\ -\frac{\alpha u_1 g}{\rho a^2} & 1 & 0 & 0 & \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{T}\delta \\ -\frac{\alpha u_2 g}{\rho a^2} & 0 & 1 & 0 & \frac{\alpha u_2}{\rho a^2}\frac{\gamma p}{T}\delta \\ -\frac{\alpha u_3 g}{\rho a^2} & 0 & 0 & 1 & \frac{\alpha u_3}{\rho a^2}\frac{\gamma p}{T}\delta \\ \frac{(\gamma-1)}{\gamma\rho}(m^2 g - 1) & 0 & 0 & 0 & 1 - (\gamma-1)m^2\delta \end{pmatrix}, \tag{5.16}$$

where

$$m^2 := \frac{\beta}{a^2} \qquad \text{and} \qquad g := 1 + (\gamma - 1)\delta.$$

To ensure that $P(\mathbf{w}_4)$ is always nonsingular we additionally require that $\beta \neq 0$. Naturally, this is only a necessary condition. But formulating a sufficient condition is not straightforward. Moreover, to check in a computation if $P(\mathbf{w}_4)$ is nonsingular yields a significant increase of complexity and is therefore unrealistic.

To investigate the effect of the preconditioner $P(\mathbf{w}_4)$ on the spectrum for low Mach numbers we need to compute the eigenvalues and corresponding eigenvectors of the operators $P(\mathbf{w}_4)B_4^{(i)}$, $i = 1, 2, 3$. To this end we transform $P(\mathbf{w}_4)$ and $B_4^{(i)}$ into a representation with respect to the entropy variables $\mathbf{w}_0$, that is we compute

$$B_0^{(i)} := \frac{\partial \mathbf{w}_0}{\partial \mathbf{w}_4} B_4^{(i)} \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_0} \qquad \text{and} \qquad P(\mathbf{w}_0) := \frac{\partial \mathbf{w}_0}{\partial \mathbf{w}_4} P(\mathbf{w}_4) \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_0}.$$

where the change of coordinates from $\mathbf{w}_0$ to $\mathbf{w}_4$ and vice versa is given by a multiplication with the matrices

$$\frac{\partial \mathbf{w}_0}{\partial \mathbf{w}_4} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1-\gamma & 0 & 0 & 0 & \frac{\gamma p}{T} \end{pmatrix} \quad \text{and} \quad \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_0} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \frac{(\gamma-1)T}{\gamma p} & 0 & 0 & 0 & \frac{T}{\gamma p} \end{pmatrix}.$$

respectively. Straightforward computations yield

$$B_0^{(1)} = \begin{pmatrix} u_1 & \rho a^2 & 0 & 0 & 0 \\ \frac{1}{\rho} & u_1 & 0 & 0 & 0 \\ 0 & 0 & u_1 & 0 & 0 \\ 0 & 0 & 0 & u_1 & 0 \\ 0 & 0 & 0 & 0 & u_1 \end{pmatrix} \quad \text{and} \quad P(\mathbf{w}_0) = \begin{pmatrix} m^2 & 0 & 0 & 0 & -m^2\delta \\ -\frac{\alpha u_1}{\rho a^2} & 1 & 0 & 0 & \frac{\alpha u_1}{\rho a^2}\delta \\ -\frac{\alpha u_2}{\rho a^2} & 0 & 1 & 0 & \frac{\alpha u_2}{\rho a^2}\delta \\ -\frac{\alpha u_3}{\rho a^2} & 0 & 0 & 1 & \frac{\alpha u_3}{\rho a^2}\delta \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Hence, in the entropy coordinates $\mathbf{w}_0$ the preconditioned operator $P(\mathbf{w}_4)B_4^{(1)}$ takes the form

$$P(\mathbf{w}_0)B_0^{(1)} = \begin{pmatrix} m^2 u_1 & \rho\beta & 0 & 0 & -m^2 u_1 \delta \\ \frac{1}{\rho}\left(1-\frac{\alpha u_1^2}{a^2}\right) & (1-\alpha)u_1 & 0 & 0 & \frac{\alpha u^2 \delta}{\rho a^2} \\ -\frac{\alpha u_1 u_2}{\rho a^2} & -\alpha u_2 & u_1 & 0 & \frac{\alpha u_1 u_2 \delta}{\rho a^2} \\ -\frac{\alpha u_1 u_3}{\rho a^2} & 0 & -\alpha u_3 & u_1 & \frac{\alpha u_1 u_3 \delta}{\rho a^2} \\ 0 & 0 & 0 & 0 & u_1 \end{pmatrix}. \tag{5.17}$$

The eigenvalues of $P(\mathbf{w}_0)B_0^{(1)}$ are given by $\Lambda_0^{(1)} = \operatorname{diag}\left(u_1, u_1, u_1, \mu_+^{(1)}, \mu_-^{(1)}\right)$ where

$$\mu_{+,-}^{(1)}(\alpha,\beta) = \frac{1}{2}\left((1-\alpha+m^2)u_1 \pm \sqrt{(1-\alpha+m^2)^2 u_1^2 + 4\beta\left(1-\frac{u_1^2}{a^2}\right)}\right)$$

and the corresponding eigenvectors are the columns of the matrix

$$U_0^{(1)} = \begin{pmatrix} 0 & 0 & 0 & \rho\beta & \rho\beta \\ 1 & 0 & \frac{\delta u}{\rho a^2} & \mu_+^{(1)} - m^2 u_1 & \mu_-^{(1)} - m^2 u_1 \\ 0 & 1 & 0 & \frac{\alpha u_2 \mu_0^4}{u_1 - \mu_0^4} & \frac{\alpha u_2 \mu_0^5}{u_1 - \mu_0^5} \\ 0 & 0 & 0 & \frac{\alpha u_3 \mu_0^4}{u_1 - \mu_0^4} & \frac{\alpha u_3 \mu_0^5}{u_1 - \mu_0^5} \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Along the lines of the computations above we conclude that the eigenvalues of $P(\mathbf{w}_0)B_0^{(i)}$, $i = 2,3$, are given by the diagonal elements of

$$\begin{aligned} \Lambda_0^{(2)} &= \operatorname{diag}\left(u_2, u_2, u_2, \mu_+^{(2)}, \mu_-^{(2)}\right) \\ \Lambda_0^{(3)} &= \operatorname{diag}\left(u_3, u_3, u_3, \mu_+^{(3)}, \mu_-^{(3)}\right) \end{aligned}$$

where

$$\mu_{+,-}^{(2)}(\alpha,\beta) = \frac{1}{2}\left((1-\alpha+m^2)u_2 \pm \sqrt{(1-\alpha+m^2)^2 u_2^2 + 4\beta\left(1-\frac{u_2^2}{a^2}\right)}\right)$$

and

$$\mu^{(3)}_{+,-}(\alpha,\beta) = \frac{1}{2}\left((1-\alpha+m^2)u_3 \pm \sqrt{(1-\alpha+m^2)^2u_3^2 + 4\beta\left(1-\frac{u_3^2}{a^2}\right)}\right).$$

Since the matrices $P(\mathbf{w}_0)B_0^{(i)}$ and $P(\mathbf{w}_4)B_4^{(i)}$, $i = 1,2,3$, are similar, their eigenvalues coincide. Obviously, it is the goal to find parameters α and β such that the eigenvalues are clustered, that is the relation

$$|\mu_+^{(i)}(\alpha_0,\beta_0)| \approx |u_i| \approx |\mu_-^{(i)}(\alpha_0,\beta_0)|, \qquad i = 1,2,3, \tag{5.18}$$

holds. Note, at the outset this problem is hard to solve since we have at least six conditions for only two unknowns. Moreover, (5.18) describes a local condition. For large-scale flow problems it is far too time consuming to determine for each cell in the grid optimal parameters α and β. It is the topic of the next section to present some parameter choice rules which can be found in the literature.

5.5 Parameter choice rules

Before we give some ideas to choose suitable parameters α,β and δ let us show some limits of the preconditioner. Obviously, it would be desirable to have the relations

$$P(\mathbf{w}_4)B_4^{(1)} = P(\mathbf{w}_4)B_4^{(2)} = P(\mathbf{w}_4)B_4^{(3)} = I. \tag{5.19}$$

From (5.19) we conclude $B_4^{(1)} = B_4^{(2)} = B_4^{(3)}$ and so the preconditioned Euler equation (5.12) simplifies to

$$\frac{\partial \mathbf{w}_4}{\partial t} + \sum_{i=1}^{3} \frac{\partial \mathbf{w}_4}{\partial x_i} = 0,$$

which is a linear advection problem and could be explicitly solved by the method of characteristics. Hence, to find a preconditioner such that (5.19) is satisfied is unrealistic and in general impossible.

Nevertheless, we can try to determine α,β and δ such that (5.19) is approximately satisfied, that is to solve the minimization problem

$$\frac{1}{2}\sum_{i=1}^{3}\left\|P(\mathbf{w}_4)B_4^{(i)} - I\right\|^2 = \min_{\alpha,\beta,\delta}!, \tag{5.20}$$

where $\|\cdot\|$ is some appropriate norm and $I = \mathrm{diag}(1,1,1,1,1)$ denotes the identity matrix. On the other hand, since we have knowledge of the eigenvalues we can solve instead of (5.20)

$$\frac{1}{2}\sum_{i=1}^{3}\left[\left(\mu_+^{(i)}(\alpha,\beta) - u_i\right)^2 + \left(\mu_-^{(i)}(\alpha,\beta) + u_i\right)^2\right] = \min_{\alpha,\beta}! \tag{5.21}$$

If α_0 and β_0 solve either (5.20) or (5.21) we expect that (5.18) is satisfied, that is the acoustic and convective waves travel approximately at the same speed. This possibly reduces the stiffness of the original system of differential equations. Note that (5.21) does not involve the determination of the parameter δ, which does not occur in the eigenvalues but only in the eigenvectors $U_0^{(i)}$.

However, the eigenvalues $\mu_{+,-}^{(i)}$, $i = 1,2,3$, depend on the velocity vector $\mathbf{u}$, which is a function of space and time variables. Hence, for large-scale flow problems described by the Euler equation (5.1) it is a complex task to compute locally optimal parameters α,β and δ. Moreover, it

is also not clear if such a locally optimal parameter choice rule would yield better convergence rates of the explicit Runge-Kutta scheme which we apply to solve (5.1). In reference [71] the authors even claim that for a similar flow problem numerous calculations showed that in general a constant choice of the parameters yields the best convergence rates. Unfortunately they could give no theoretical reasons for their observation.

To this end let us present some heuristic parameter choice rules which can be found in the literature (see e.g. [57]). These rules usually try to satisfy three main aspects:

a) For low Mach numbers the parameters α, β and δ need to be chosen such that the eigenvalues of $\Lambda_0^{(i)}$, $i = 1,2,3$ are of about the same magnitude.

b) For transonic and supersonic flows the parameters α, β and δ should be chosen such that the preconditioner has only little influence on the system or preconditioning is even turned off.

c) To ensure that the preconditioner $P(\mathbf{w}_4)$ is nonsingular it is a necessary condition that $m^2 > 0$. Moreover, to avoid instabilities it is even required to bound m^2 away from zero, that is

$$m^2 = \frac{\beta}{a^2} > \varepsilon > 0$$

for some $\varepsilon > 0$.

Let us start with a choice of parameters to satisfy condition b). To simplify the situation we choose $\alpha = 0$. Then we need to determine δ and β such that

$$\begin{aligned} \beta(1+(\gamma-1)\delta) &= a^2, && \text{(5.22a)} \\ \beta\frac{\gamma p}{T}\delta &= 0, && \text{(5.22b)} \\ (\gamma-1)\delta &= 0. && \text{(5.22c)} \end{aligned}$$

From (5.22b) and (5.22c) we conclude that for supersonic flows $\delta = 0$ is required to turn the preconditioning off. Then, by (5.22a) we have $\beta = a^2$. Indeed, for this parameter choice $P(\mathbf{w}_4) = I$, that is preconditioning is turned off.

In the low Mach number case our goal is to minimize the relations

$$\left|\frac{\mu_+^{(i)}}{\mu_-^{(i)}}\right|, \qquad i = 1,2,3.$$

Note that this problem is independent of δ. Since we are in the low Mach number case, that is $M \approx 0$, we neglect terms which contain the Mach number $M := \|\mathbf{u}\|_2/a$. If we assume that β is a function of the flow velocity, that is

$$\beta := \beta(\mathbf{u}) := \|\mathbf{u}\|_2^2$$

we have

$$\begin{aligned} \mu_+^{(i)} &= \frac{1}{2}\left((1-\alpha)u_i + \sqrt{(1-\alpha)^2u_i^2+4\beta}\right) \leq \frac{1}{2}\left((1-\alpha)u_i + \sqrt{(1-\alpha)^2u_i^2+4}\right) \\ \mu_-^{(i)} &= \frac{1}{2}\left((1-\alpha)u_i - \sqrt{(1-\alpha)^2u_i^2+4\beta}\right) \geq \frac{1}{2}\left((1-\alpha)u_i - \sqrt{(1-\alpha)^2u_i^2+4}\right) \end{aligned}$$

and therefore for $\alpha = 0$ we have

$$\left|\frac{\mu_+^{(i)}}{\mu_-^{(i)}}\right| \approx \left|\frac{1+\sqrt{5}}{1-\sqrt{5}}\right|, \qquad i = 1,2,3. \tag{5.23}$$

Unfortunately, so far we did not take care of condition c). To avoid instabilities in [15] it was suggested to let β not only depend on $\mathbf{u}$, but also to define β piecewise, more precisely

$$\beta(\mathbf{u}) = \begin{cases} \varepsilon, & M < \varepsilon, \\ \|\mathbf{u}\|_2^2, & \varepsilon \le M < 1, \\ a^2, & M \ge 1, \end{cases} \tag{5.24}$$

for some $\varepsilon > 0$. In this context ε plays the role of some threshold parameter. Unfortunately at the outset it is not clear how to determine a suitable ε. For example, choosing it too small possibly leads to numerical instabilities.

In [57] the following choice for β, which is only slightly different from (5.24), was suggested,

$$\beta(\mathbf{u}) = \min\left\{\max\left\{\|\mathbf{u}\|_2^2, K\|\mathbf{u}_\infty\|_2^2\right\}, a^2\right\} \tag{5.25}$$

or rather

$$\beta(\mathbf{u}) = \min\left\{\max\left\{\|\mathbf{u}\|_2^2\left(1+\frac{(1-M_0^2)}{M_0^4}M^2\right), K\|\mathbf{u}_\infty\|_2^2\right\}, a^2\right\}. \tag{5.26}$$

Here $\|\mathbf{u}_\infty\|_2$ denotes the speed of the inflowing fluid. The choice (5.25) has the effect that for supersonic flow $\beta = a^2$. As mentioned above, this is a desirable effect since preconditioning should be turned off for supersonic flows. The parameter K needs to be set a-priori and plays – as above the ε – the role of a cutoff value. In numerical applications the choice $K = 1$ is often a suitable choice and yields sufficient convergence results. If this choice leads to instabilities numerical experience has shown that one should try choose $K \in [1,4]$.

Finally, let us summarize which parameter choice rules are implemented in the DLR TAU-Code. Since the effect of α is harder to analyze and not so well understood throughout all implemented methods $\alpha = 0$. To determine β we use criterion (5.25). For the choice of δ two different implementations are available.

a) The first one is a hard coded $\delta = 1$. This implementation worked quite well over the last several years but it often leads to numerical instabilities in mixed flow fields where there are regions of subsonic and supersonic flows.

b) Instead of a hard coded δ we distinguish whether the current cell-center is in a subsonic or supersonic state. To this end we define $\delta := \delta(M)$ where

$$\delta(M) := \begin{cases} 0, & M^2 \ge 1, \\ 1, & M^2 < 1. \end{cases} \tag{5.27}$$

Numerical experience has shown that this choice often yields to a better stability of the algorithm. Moreover, if the user forgets to turn preconditioning off for a transonic or supersonic case often numerical instabilities can be avoided.

Naturally, we can easily implement other choices for the parameters. For example, instead of choosing for δ the cutoff function (5.27) we could also think of any (continuous) function δ satisfying $\delta(M) = 0$, $M \ge 1$ and $\delta(0) = 1$, for instance

$$\delta(M) := \begin{cases} 0, & M^2 \ge 1, \\ f(M), & 0 \le M^2 < 1, \end{cases}$$

where $f(t) = 1 - t^n$. Further experience and research is required to optimize the parameter n. The choice (5.27) corresponds to the case $n = \infty$.

Naturally, there is also potential to optimize the parameters α and β. For example, instead of using fixed values for β by (5.24) it is indicated to determine β also by some function of the flow velocity. Still for the parameters δ and β at least their maximum and minimum value are determined, namely

$$\beta \in \left[\varepsilon, a^2\right] \qquad \text{and} \qquad \delta \in [0,1],$$

where ε is determined by either (5.24) or (5.25) or (5.26). An appropriate choice of α is so far not clear at all. Hence, in particular it would be interesting to investigate the choice of α in future work.

5.6 Implemented preconditioners

In the DLR TAU-Code basically one preconditioning method for explicit Runge-Kutta methods is available to compute steady state solutions of the Euler and Navier Stokes equation. This method is implemented in three different ways. It is the topic of this section to compute explicit formulas of the preconditioner. These explicit formulas are required for an efficient implementation.

The first implemented preconditioning technique is based on the conservative variables $\mathbf{w}_1$. Then the preconditioner Γ given by (5.9) takes the form

$$\Gamma(\mathbf{w}_1, \mathbf{w}_4) = \frac{\partial \mathbf{w}_1}{\partial \mathbf{w}_4} P(\mathbf{w}_4) \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}$$

and the preconditioned Euler equation (5.10) simplifies to

$$\frac{\partial \mathbf{w}_1}{\partial t} + \frac{\partial \mathbf{w}_1}{\partial \mathbf{w}_4} P(\mathbf{w}_4) \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1} \sum_{i=1}^{3} \frac{\partial \mathbf{f}^{(i)}}{\partial x_i} = 0, \tag{5.28}$$

that is for an efficient implementation an explicit formula for $\Gamma(\mathbf{w}_1, \mathbf{w}_4)$ is required. Note that for large scale flow problems it is in particular desirable to avoid the two matrix multiplications required to determine

$$\frac{\partial \mathbf{w}_1}{\partial \mathbf{w}_4} P(\mathbf{w}_4) \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}, \tag{5.29}$$

since these multiplications yield a significant increase of the total complexity.

Experience shows that preconditioning of the conservative variables $\mathbf{w}_1$ yields more often to a loss of stability when compared to preconditioning of the primitive variables $\mathbf{w}_4$. The reasons for this observation are not clear to us. However, with respect to the primitive variables $\mathbf{w}_4$ the preconditioner Γ is represented through $\Gamma(\mathbf{w}_4, \mathbf{w}_4)$ given by

$$\Gamma(\mathbf{w}_4, \mathbf{w}_4) = P(\mathbf{w}_4) \frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1}.$$

To this end, the preconditioned Euler equation (5.10) with respect to the primitive variables is also given by (5.28). Hence, an explicit formula for the matrix (5.29) is required. To this end let us now explicitly compute the entries of $\Gamma(\mathbf{w}_4, \mathbf{w}_4)$. Note, we do not compute explicity every

entry. The missing entries follow by symmetry arguments. For the first row we compute using

$$\frac{\partial \mathbf{w}_4}{\partial \mathbf{w}_1} = \begin{pmatrix} \frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} & (1-\gamma)u_1 & (1-\gamma)u_2 & (1-\gamma)u_3 & \gamma-1 \\ -\frac{u_1}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{u_2}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{u_3}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ \frac{1}{\rho}\left(\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - T\right) & \frac{(1-\gamma)u_1}{\rho} & \frac{(1-\gamma)u_2}{\rho} & \frac{(1-\gamma)u_3}{\rho} & \frac{\gamma-1}{\rho} \end{pmatrix}$$

$$\begin{aligned}
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{1,1} &= m^2 g\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - m^2\frac{\gamma p}{T}\delta\frac{1}{\rho}\left(\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - T\right) \\
&= m^2(1+\delta(\gamma-1))\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - m^2a^2\delta\left(\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2T} - 1\right) \\
&= \frac{(\gamma-1)m^2\|\mathbf{u}\|_2^2}{2}\left(1+\delta\gamma-\delta-\frac{a^2\delta}{T}\right) + \beta\delta \\
&= \beta\delta - (\gamma-1)(\delta-1)\frac{m^2\|\mathbf{u}\|_2^2}{2}, \\
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{1,2} &= m^2 g(1-\gamma)u_1 - m^2\frac{\gamma p}{T}\delta(1-\gamma)\frac{u_1}{\rho} \\
&= m^2u_1(1-\gamma)(g-\gamma\delta) \\
&= (\gamma-1)(\delta-1)m^2u_1, \\
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{1,3} &= m^2 g(1-\gamma)u_2 - m^2\frac{\gamma p}{T}\delta\frac{(1-\gamma)u_2}{\rho} \\
&= (1-\gamma)m^2u_2(g-\gamma\delta) \\
&= (\gamma-1)(\delta-1)m^2u_2 \\
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{1,5} &= m^2 g(\gamma-1) - m^2\frac{\gamma p}{T}\delta\frac{\gamma-1}{\rho} \\
&= m^2(\gamma-1)(g-\gamma\delta) \\
&= -(\gamma-1)(\delta-1)m^2,
\end{aligned}$$

and for the second row

$$\begin{aligned}
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{2,1} &= -\frac{\alpha u_1 g}{\rho a^2}\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - \frac{u_1}{\rho} + \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{T}\frac{\delta}{\rho}\left(\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - T\right)\\
&= -\frac{u_1}{\rho} - \frac{\alpha u_1(1+(\gamma-1)\delta)}{\rho a^2}\frac{(\gamma-1)\|\mathbf{u}_2\|^2}{2} + \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{T}\frac{\delta}{\rho}\left(\frac{(\gamma-1)\|\mathbf{u}\|_2^2}{2} - T\right)\\
&= -\frac{u_1}{\rho} + (\gamma-1)(\delta-1)\frac{\alpha u_1\|\mathbf{u}\|_2^2}{2\rho a^2} - \frac{\alpha u_1}{\rho a^2}\gamma\delta T\\
&= -\frac{u_1}{\rho} + (\gamma-1)(\delta-1)\frac{\alpha u_1\|\mathbf{u}\|_2^2}{2\rho a^2} - \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{\rho}\delta\\
&= -\frac{u_1}{\rho}(1+\alpha\delta) + (\gamma-1)(\delta-1)\frac{\alpha u_1\|\mathbf{u}\|_2^2}{2\rho a^2}\\
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{2,2} &= -\frac{\alpha u_1 g}{\rho a^2}(1-\gamma)u_1 + \frac{1}{\rho} + \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{T}\delta\frac{(1-\gamma)u_1}{\rho}\\
&= \frac{1}{\rho} + \frac{\alpha u_1}{\rho a^2}\left(\gamma\delta(1-\gamma) - (1+(\gamma-1)\delta)(1-\gamma)\right)\\
&= \frac{1}{\rho} - (\gamma-1)(\delta-1)\frac{\alpha u_1^2}{\rho a^2},\\
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{2,3} &= -\frac{\alpha u_1 g}{\rho a^2}(1-\gamma)u_2 + \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{T}\delta\frac{(1-\gamma)u_2}{\rho}\\
&= \frac{\alpha u_1 u_2}{\rho a^2}(1-\gamma)\left(\gamma\delta - (1+\gamma\delta-\delta)\right)\\
&= -(\gamma-1)(\delta-1)\frac{\alpha u_1 u_2}{\rho a^2}\\
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{2,5} &= \frac{-\alpha u_1 g}{\rho a^2}(\gamma-1) + \frac{\alpha u_1}{\rho a^2}\frac{\gamma p}{T}\delta\frac{(\gamma-1)}{\rho}\\
&= (\gamma-1)(\delta-1)\frac{\alpha u_1}{\rho a^2}.
\end{aligned}$$

Note that the missing entries for the first, second, third and fourth row follow by symmetry arguments. Concluding we want to compute one element from the fifth row,

$$\begin{aligned}
\Gamma(\mathbf{w}_4,\mathbf{w}_4)_{5,2} &= \frac{(\gamma-1)}{\gamma\rho}(m^2 g - 1)(1-\gamma)u_1 + (1-(\gamma-1)m^2\delta)\frac{(1-\gamma)u_1}{\rho}\\
&= \frac{(1-\gamma)u_1}{\gamma\rho}\left((\gamma-1)(m^2(1+(\gamma-1)\delta)-1) + \gamma - \gamma(\gamma-1)m^2\delta\right)\\
&= \frac{(1-\gamma)u_1}{\gamma\rho}\left(1-(\gamma-1)(\delta-1)m^2\right).
\end{aligned}$$

Hence, summarizing the computations above we have

$$\begin{aligned}\Gamma(\mathbf{w}_4,\mathbf{w}_4) \;:=\; & \begin{pmatrix} \beta\delta & 0 & 0 & 0 & 0 \\ -\frac{u_1}{\rho}(1+\alpha\delta) & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{u_2}{\rho}(1+\alpha\delta) & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{u_3}{\rho}(1+\alpha\delta) & 0 & 0 & \frac{1}{\rho} & 0 \\ \frac{\gamma-1}{\gamma\rho}\left(\beta\delta+\|\mathbf{u}\|_2^2-H\right) & \frac{(1-\gamma)u_1}{\gamma\rho} & \frac{(1-\gamma)u_2}{\gamma\rho} & \frac{(1-\gamma)u_3}{\gamma\rho} & \frac{\gamma-1}{\gamma\rho} \end{pmatrix} \\ + \quad \theta & \begin{pmatrix} -\frac{m^2\|\mathbf{u}\|_2^2}{2} & m^2u_1 & m^2u_2 & m^2u_3 & -m^2 \\ \frac{\alpha u_1\|\mathbf{u}\|_2^2}{2a^2\rho} & -\frac{\alpha u_1^2}{a^2\rho} & -\frac{\alpha u_1u_2}{a^2\rho} & -\frac{\alpha u_1u_3}{a^2\rho} & \frac{\alpha u_1}{a^2\rho} \\ \frac{\alpha u_2\|\mathbf{u}\|_2^2}{2a^2\rho} & -\frac{\alpha u_1u_2}{a^2\rho} & -\frac{\alpha u_2^2}{a^2\rho} & -\frac{\alpha u_2u_3}{a^2\rho} & \frac{\alpha u_2}{a^2\rho} \\ \frac{\alpha u_3\|\mathbf{u}\|_2^2}{2a^2\rho} & -\frac{\alpha u_1u_3}{a^2\rho} & -\frac{\alpha u_2u_3}{a^2\rho} & -\frac{\alpha u_3^2}{a^2\rho} & \frac{\alpha u_3}{a^2\rho} \\ -\frac{(\gamma-1)m^2}{\gamma\rho}\frac{\|\mathbf{u}\|_2^2}{2} & \frac{(\gamma-1)m^2u_1}{\gamma\rho} & \frac{(\gamma-1)m^2u_2}{\gamma\rho} & \frac{(\gamma-1)m^2u_3}{\gamma\rho} & -\frac{(\gamma-1)m^2}{\gamma\rho} \end{pmatrix}\end{aligned}$$

where

$$\theta := (\delta-1)(\gamma-1).$$

Finally, for the implemented case $\alpha = 0$ we get for (5.29) using

$$\frac{\partial \mathbf{w}_1}{\partial \mathbf{w}_4} = \begin{pmatrix} \frac{\rho}{p} & 0 & 0 & 0 & -\frac{\rho}{T} \\ \frac{\rho u_1}{p} & \rho & 0 & 0 & -\frac{\rho u_1}{T} \\ \frac{\rho u_2}{p} & 0 & \rho & 0 & -\frac{\rho u_2}{T} \\ \frac{\rho u_3}{p} & 0 & 0 & \rho & -\frac{\rho u_3}{T} \\ \frac{E}{p} & \rho u_1 & \rho u_2 & \rho u_3 & -\frac{\rho\|\mathbf{u}\|_2^2}{2T} \end{pmatrix}$$

the explicit formula

$$\frac{\partial \mathbf{w}_1}{\partial \mathbf{w}_4}\Gamma(\mathbf{w}_4,\mathbf{w}_4) = \begin{pmatrix} 1+\zeta_1 & \zeta_2u_1 & \zeta_2u_2 & \zeta_2u_3 & -\zeta_2 \\ \zeta_1u_1 & 1+\zeta_2u_1^2 & \zeta_2u_1u_2 & \zeta_2u_1u_3 & -\zeta_2u_1 \\ \zeta_1u_2 & \zeta_2u_1u_2 & 1+\zeta_2u_2^2 & \zeta_2u_2u_3 & -\zeta_2u_2 \\ \zeta_1u_3 & \zeta_2u_1u_3 & \zeta_2u_2u_3 & 1+\zeta_2u_3^2 & -\zeta_2u_3 \\ \frac{a^2\zeta_0\zeta_1}{\gamma-1} & \zeta_0\zeta_3u_1 & \zeta_0\zeta_3u_2 & \zeta_0\zeta_3u_3 & 1-\zeta_0\zeta_3 \end{pmatrix}$$

where

$$\begin{aligned}\zeta_3 &:= 1+(\delta-1)\frac{\beta}{a^2} \\ \zeta_2 &:= \frac{\gamma-1}{a^2}\zeta_3 \\ \zeta_1 &:= \frac{\beta}{a^2}\delta-\frac{\|\mathbf{u}\|_2^2}{2}\zeta_2 \\ \zeta_0 &:= 1+\frac{\gamma-1}{2}M^2.\end{aligned}$$

Note, if in a supersonic state we have by (5.25) $\beta = a^2$ and by (5.27) $\delta = 0$ and hence $\zeta_3 = \zeta_2 = \zeta_1 = 0$, then

$$\frac{\partial \mathbf{w}_1}{\partial \mathbf{w}_4}\Gamma_4 = I \in \mathbb{R}^{5\times 5},$$

that is (5.10) simplifies to the non-preconditioned Euler equation (5.1). This yields numerical stability of preconditioning in the case of larger Mach numbers. Due to (5.11) we still need an explicit formula for $\Gamma(\mathbf{w}_4,\mathbf{w}_4)^{-1}$ for $\alpha = 0$, which is given by

$$\Gamma(\mathbf{w}_4,\mathbf{w}_4)^{-1} = \begin{pmatrix} \frac{1}{m^2}-\sigma_1 & 0 & 0 & 0 & \sigma_1\frac{\gamma p}{\gamma-1} \\ \left(\frac{1}{m^2}-\sigma_1\right)u_1 & \rho & 0 & 0 & \sigma_1\frac{\gamma p}{\gamma-1}u_1 \\ \left(\frac{1}{m^2}-\sigma_1\right)u_2 & 0 & \rho & 0 & \sigma_1\frac{\gamma p}{\gamma-1}u_2 \\ \left(\frac{1}{m^2}-\sigma_1\right)u_3 & 0 & 0 & \rho & \sigma_1\frac{\gamma p}{\gamma-1}u_3 \\ \frac{H}{m^2}-1-\sigma_2 & \rho u_1 & \rho u_2 & \rho u_3 & (1+\sigma_2)\frac{\gamma p}{\gamma-1} \end{pmatrix},$$

where

$$\sigma_1 := (\delta-1)\frac{\gamma-1}{a^2} \qquad \text{and} \qquad \sigma_2 := (\delta-1)\left(1+\frac{\gamma-1}{2}M^2\right).$$

For the rest of this chapter we refer to the different implementations of preconditioning based on the different parameter choice rules in the following way:

a) With Γ^{Pold} we denote the implementation of the preconditioner with respect to the primitive variables and parameters $\alpha = 0$, $\delta = 1$ and β given by (5.25).

b) With Γ^{Pnew} we denote the implementation of the preconditioner with respect to the primitive variables and parameters $\alpha = 0$, δ given by (5.27) and β by (5.25).

c) With Γ^{Pcons} we denote the implementation of the preconditioner with respect to the conservative variables and parameters $\alpha = 0$, $\delta = 0$ and β given by (5.25).

5.7 Numerical examples

Although the theory in this chapter has been only carried out for simplicity for the Euler equation, preconditioning also works pretty good for Navier-Stokes test cases. To validate the different methods we plot the residuals, the pressure drag C-drag and the pressure coefficient cp. The pressure drag can be used as a measure of accuracy because for subsonic inviscid flow it vanishes. Finally note that because of (5.23) we expect that in the preconditioned simulations the convergence speed and the accuracy is independent of the onflow Mach number.

5.7.1 NACA0012

We start our numerical investigations of the implemented preconditioning techniques with the flow over a NACA0012 airfoil. In Figure 5.2 a section of the Euler grid used for the computations is plotted. First of all, by Figure 5.3 it can be observed that for low Mach numbers the convergence deteriorates rapidly. This loss of convergence and accuracy can be avoided by applying preconditioning. Figures 5.4, 5.5 and 5.6 show that Γ^{Pold} and Γ^{Pnew} yield similar results.

As indicated in Section 5.5 from Figures 5.8, 5.9 and 5.10 we observe the superiority of the choice of δ by (5.27) when compared with the hard coded $\delta = 1$. For all Mach numbers $M \in \{0.75, 0.8, 1.2\}$ the preconditioned flow solver shows instabilities (see Figure 5.9) for the hard coded δ whereas in the case where the preconditioner takes care of supersonic and subsonic

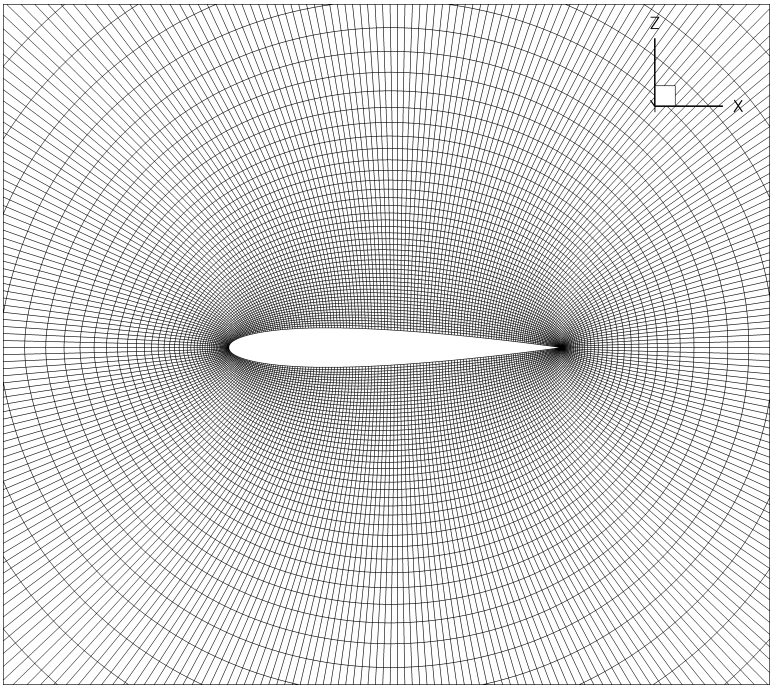


Figure 5.2: NACA0012 Euler grid

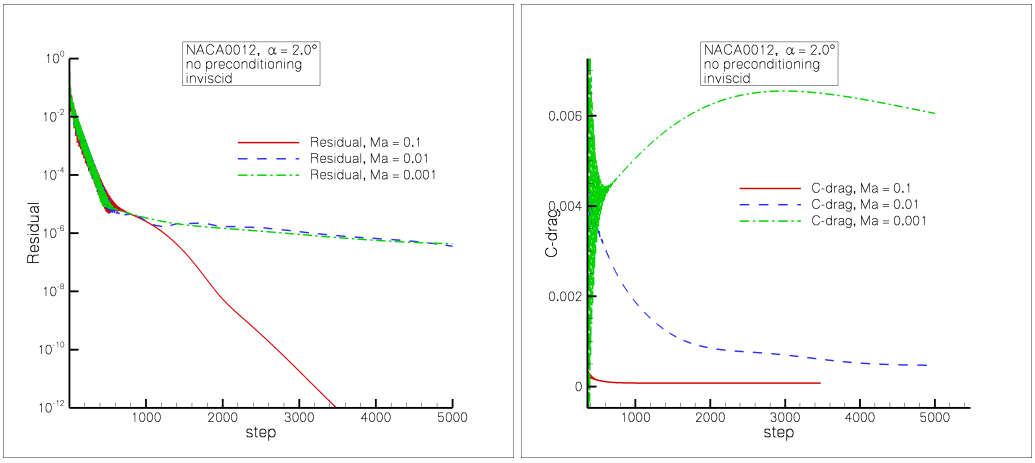


Figure 5.3: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers without preconditioning

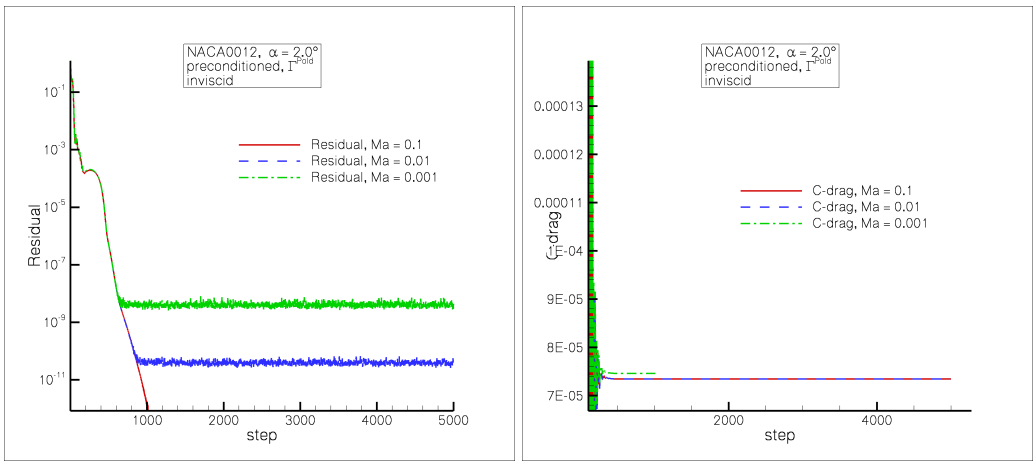


Figure 5.4: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pold}

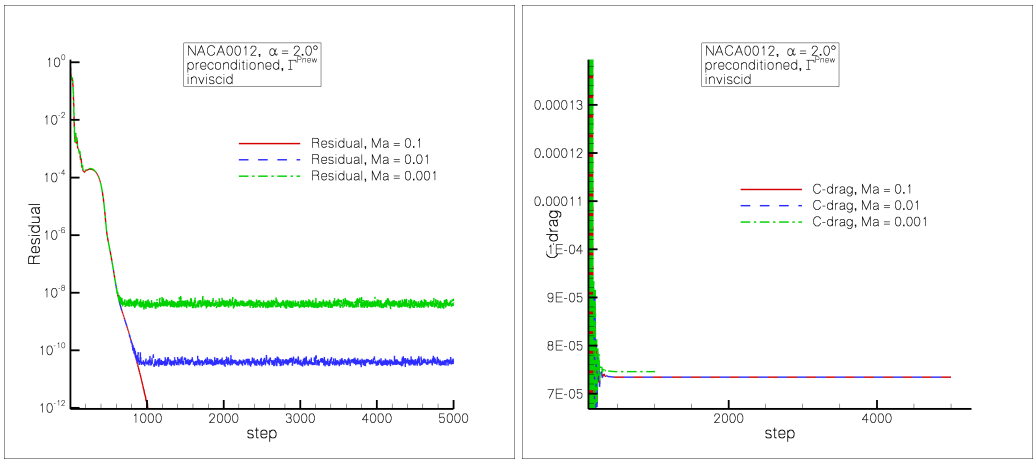


Figure 5.5: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pnew}

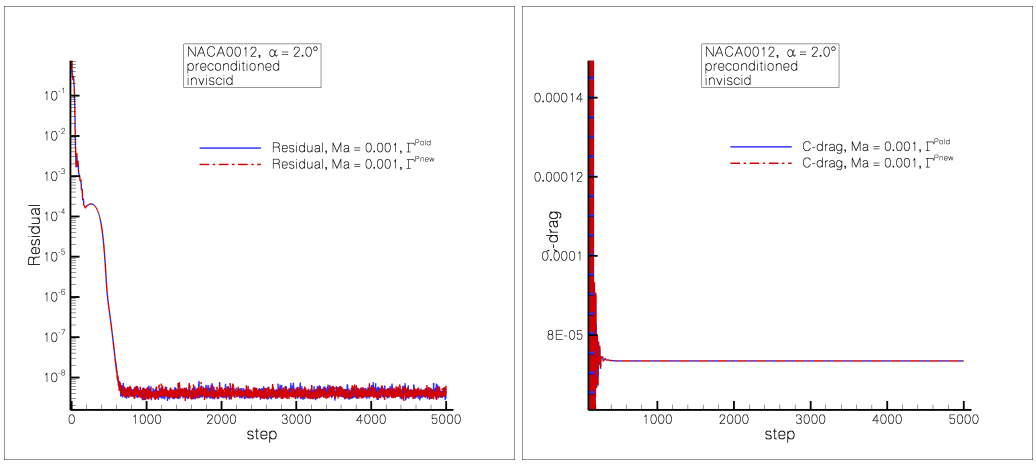


Figure 5.6: NACA0012 airfoil, angle of attack 2.0°, comparison of the residuals and C-drag for Mach number 0.001 with preconditioning

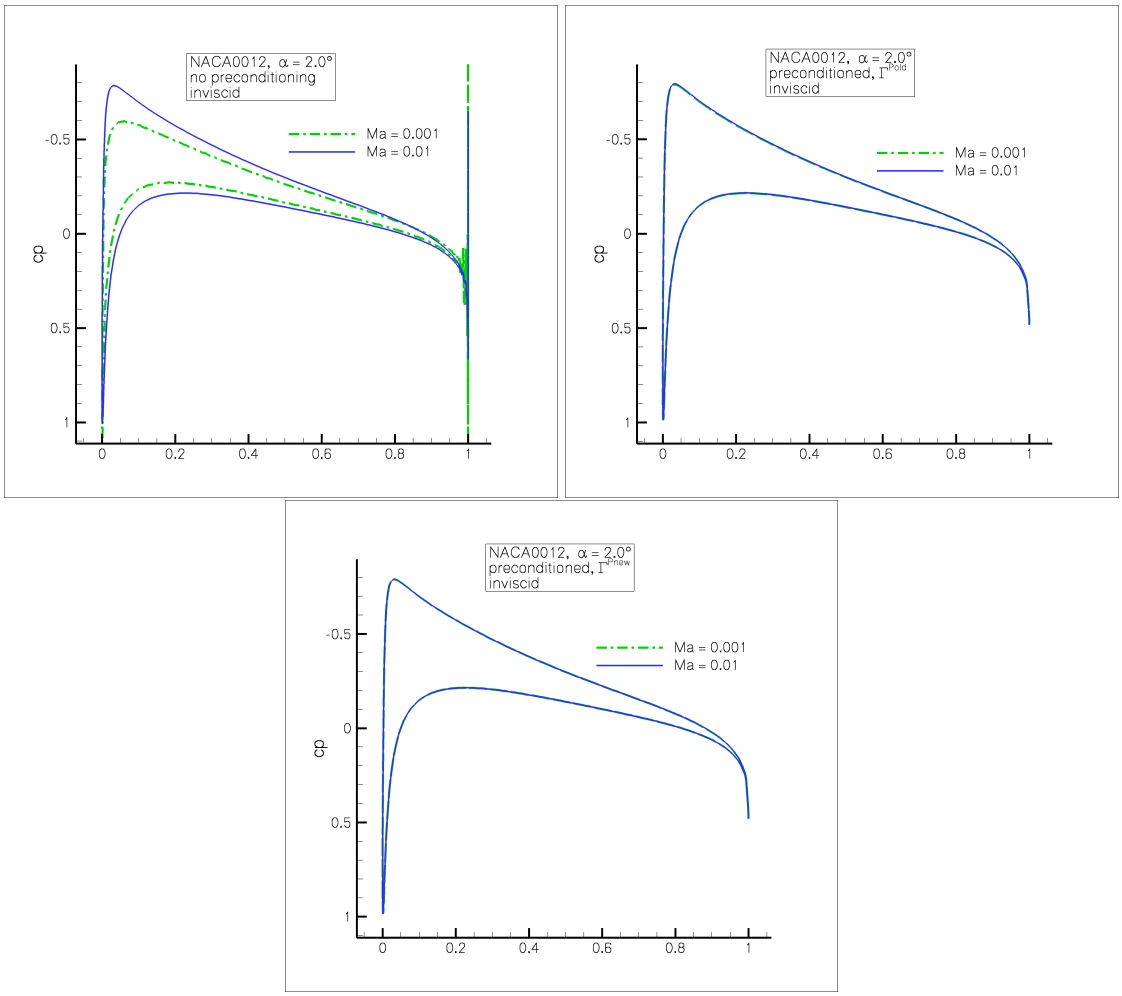


Figure 5.7: NACA0012 airfoil, angle of attack 2.0°, cp for Mach numbers 0.01 and 0.001 without preconditioning, with preconditioning Γ^{Pold} and preconditioning Γ^{Pnew}

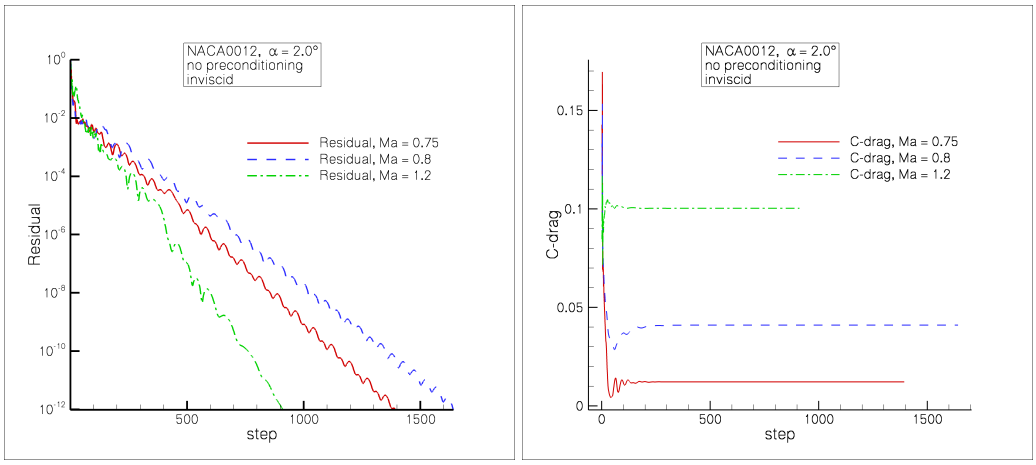


Figure 5.8: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers without preconditioning

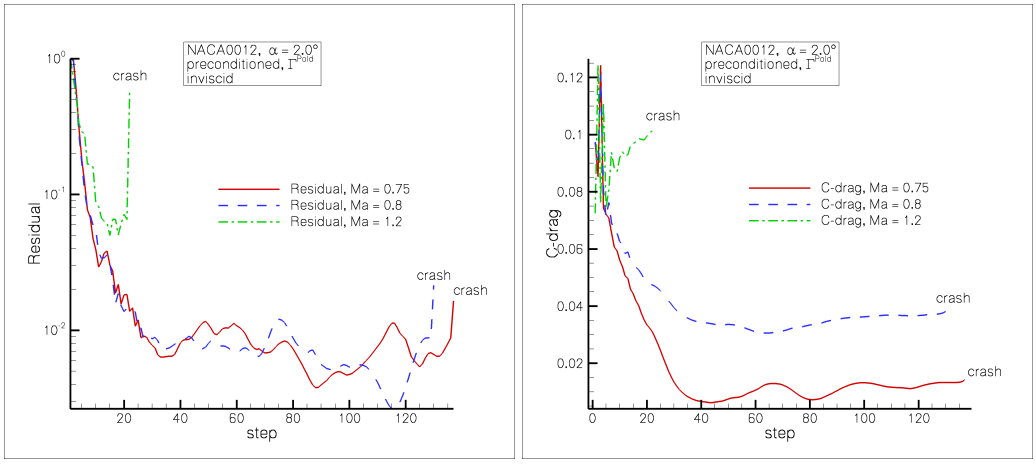


Figure 5.9: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pold}

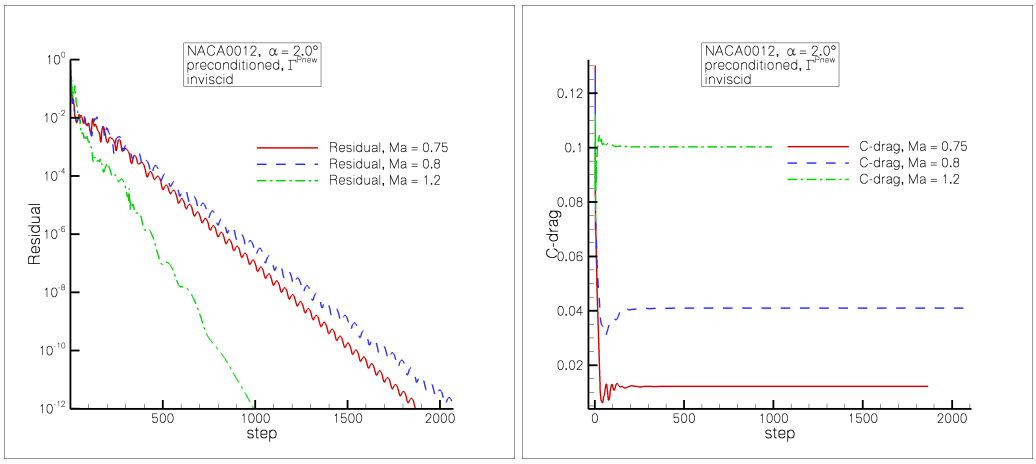


Figure 5.10: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pnew}

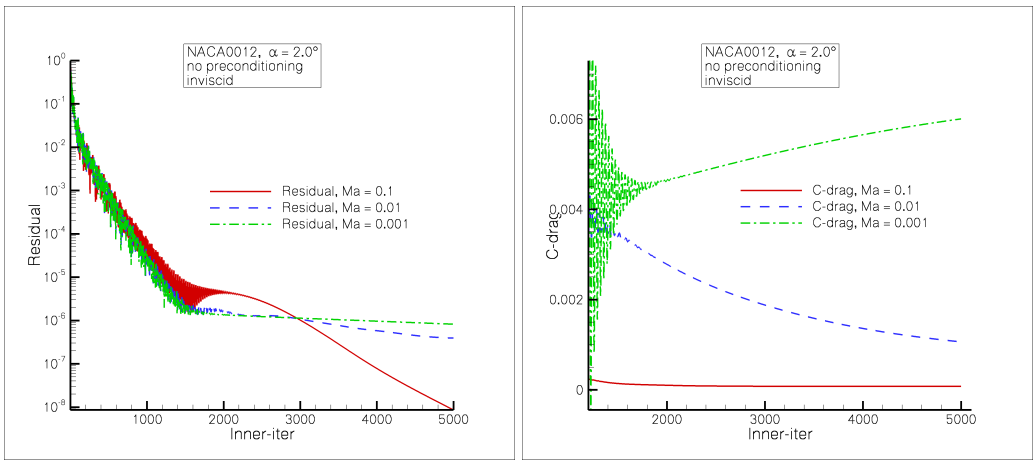


Figure 5.11: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers and LUSGS-scheme without preconditioning

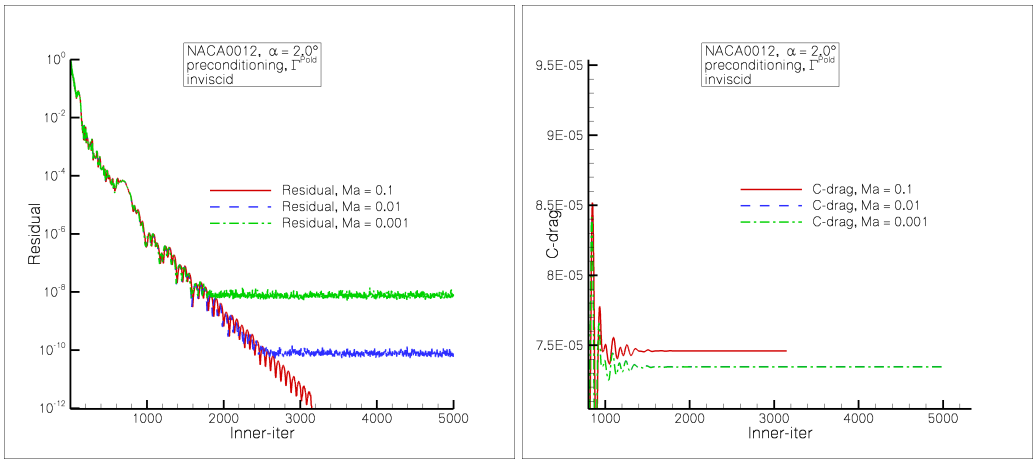


Figure 5.12: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers and LUSGS-scheme with preconditioning Γ^{Pold}

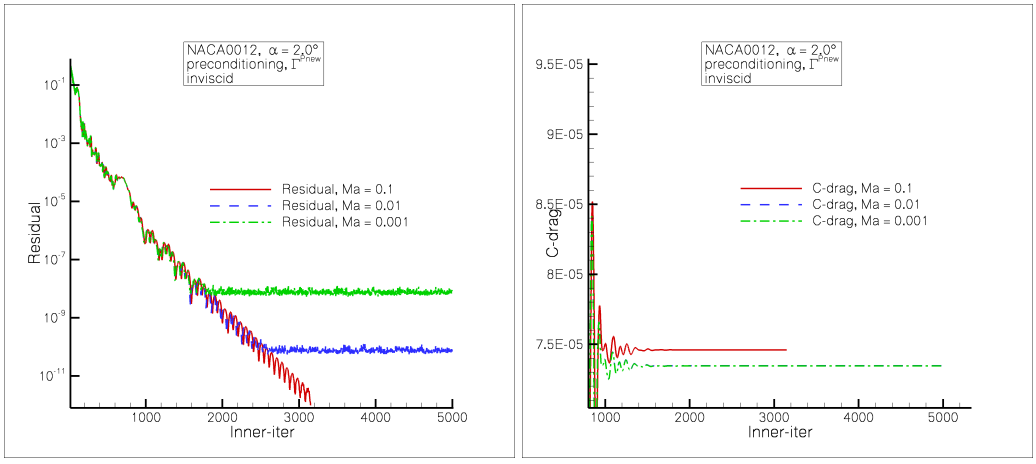


Figure 5.13: NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers and LUSGS-scheme with preconditioning Γ^{Pnew}

flows the results are comparable to those without preconditioning. Unfortunately the convergence speed with turned on preconditioning Γ^{Pnew} is slower when compared with preconditioning turned off. In Figure 5.7 we have plotted the cp distribution on the airfoil. Once more it can be observed that in the preconditioned case the obtained accuracy is independent of the Mach number. accuracy. Finally, Figures 5.11 – 5.13 show some results we obtained by using a LUSGS-scheme. For this method the superiority of preconditioning is also evident.

5.8 Technical implementation details

In this section we shortly present some technical details of changes in the DLR TAU-Code with respect to the new implemented parameter choice rules for preconditioning. We start with the user interface.

The preconditioning techniques implemented so far could be chosen in the following way. In the parameter-file parameters were defined through:

```
Preconditioner(0/1/2)
Cut-off value
```

The choice `0` was the default value and implied that preconditioning is turned off, `1` turned the preconditioning with respect to the primitive variables on and `2` turned the preconditioning with respect to the conservative variables on. The default value of the `Cut-off value` is 1. The `Cut-off value` corresponds to the parameter K in (5.25).

Since TAU release 2008.1.0 the possibilities of preconditioning will be extended and can be chosen in the following way. In the parameter-file parameters are defined through

```
Preconditioner
Cut-off value
```

The `Cut-off value` works as described above. The valid parameters for the parameter `Preconditioner` have been replaced by

```
(none)
PrimOld
PrimNew
Conservative
```

Naturally, `(none)` means that preconditioning is turned off, which is the default value. The preconditioning methods are based on the preconditioner $\Gamma(\mathbf{w}_4, \mathbf{w}_4)$ and differ on the choice of the parameters. For the TAU-user only methods for the choice $\alpha = 0$ are available. The choice `PrimOld` represents preconditioning based on the primitive variables, $\delta = 1$ is hard coded and β given by (5.25). The suffix `Old` means that this method coincides with the preconditioning method which was already available in older TAU-versions. This method corresponds to the case `1` in the older versions.

The choice `PrimNew` represents preconditioning based on the primitive variables, δ given by (5.27) and β given by (5.25). The suffix `New` means that this method is not available in older TAU-versions.

The method `Conservative` corresponds to the case `2` in the older versions. Supported by the numerical examples shown in Section 5.7 we recommend to use `PrimNew`.

Besides the methods above we implemented the general preconditioner $\Gamma(\mathbf{w}_4, \mathbf{w}_4)$, which is only available in the developer version of the code. This is due to the fact that further research is necessary for an appropriate choice of the parameters α, β and δ to speed up convergence. So

far, unfortunately, we were not able to find satisfactory parameter choices. On some examples we could already show that better convergence rates are available, but in other examples these parameters yield numerical instabilities.

Moreover, so far the preconditioning had been implemented in three different files of the TAU-Code,

```
residuals.c
flux_classic_dissipation.c          .
flux_central_dissipation_reciproc.c
```

To improve the quality of the TAU-Code we have removed the code with respect to the preconditioning from these files to the already existing files `preconditioner.h` and `preconditioner.c`. In `preconditioner.c` the different preconditioners and their inverse are now implemented. This has the advantage that in future work other available preconditioning techniques or parameter choice rules can be easily added into the TAU-Code.

5.9 Concluding remarks

Finally, let us shortly summarize our results and give an outlook.

The new implemented preconditioning technique in the DLR TAU-Code showed in all the examples we considered superiority to the old method. This is mainly due to the fact that the more flexible parameter choice rule avoids in particular for large Mach numbers numerical instabilities.

Because of this fact we think that further research for better suited parameter choices can be profitable to speed up the explicit Runge-Kutta methods applied to compute steady state solutions. In particular a suitable choice of the parameter α is an open problem.

6 Parallelization

6.1 Introduction

For parallel computing on grids (especially unstructured grids) domain decomposition is a powerful concept: Given a number P of processors, the whole computational domain (grid) is decomposed into P subdomains (subgrids). Each of the P processors computes on one of the subgrids.

Usually communication between the processors is required because the solutions on the subdomains depend on each other. Here the algorithms required for the parallelization of the solver are described. In the description variable names are used to make references to the data as simple as possible. Variable names are typeset in a style like **var**. The notation of structures are used like they are defined in the C programming language for containers of data. The notation is **containervar.contentsvar**, where **containervar** is a variable of container type (C struct) containing several variables **contentsvar**. Note that **contentsvar** alone is not defined.

Furthermore the convention that arrays are indexed from 0 to $\dim-1$ is used like in C and that we can assign complete arrays (what we cannot in C).

6.2 Specification of the domain decomposition

The solver operates in different ways on two in principle different kinds of data types:

1. Operations that compute point variables directly from one or more point variables. For these operations no data of other points is required to compute on one point.

2. Operations that need data of some *neighboring* points to compute the result for one point. For these operations connectivity information is required. The main kind of connectivity data used in the solver are the edges. Other connectivity data is given by the boundary faces where a near point is connected to a boundary point. More connectivity data is defined by the grid to grid connections of the different gridlevels from the multigrid algorithm.

As much as possible of these operations should be performed on one subdomain of the grid without communication. Furthermore the required communication interface should be clear and simple to ensure that further development of the flow-solver keeps as easy as possible without re-investigating parallelization.

6.2.1 Assignment to the subdomains

The data is assigned as follows to the subdomains:

1. Each point of the full grid is assigned uniquely to one subdomain. We say that these points are owned by the corresponding subdomains.

2. All connectivity data that is required to compute a sub-result for the own points is assigned to the subdomains.

3. All points referenced by the connectivity data on a subdomain that are not owned by it are also assigned to the subdomain. These points are called additional points.

Example

The following example helps to make the assignment clearer.

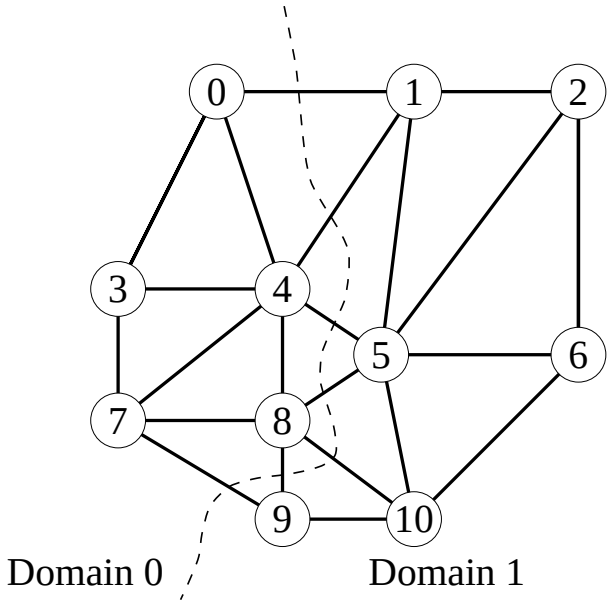


Figure 6.1: Triangulation, sub-divided into two subdomains

The Figure 6.1 illustrates: the triangulation is built from 11 points from which 8 are located on the boundary, the connectivity is given by 22 edges. In the first step five points (0, 3, 4, 7 and 8) are assigned to domain 0, the remaining six points (1, 2, 5, 6, 9 and 10) are assigned to domain 1.

The connectivity data for the five own points of domain 0 is given by the following 14 edges: 0-1, 0-3, 0-4, 1-4, 3-4, 3-7, 4-5, 4-7, 4-8, 5-8, 7-8, 7-9, 8-9 and 8-10. When inspecting these edges one finds that in addition to the five own points (0, 3, 4, 7 and 8) also four additional points (1, 5, 9 and 10) are referenced. These additional points are also needed on domain 0.

On domain 1 15 edges are needed: 0-1, 1-2, 1-4, 1-5, 2-5, 2-6, 4-5, 5-6, 5-8, 5-10, 6-10, 7-9, 8-9, 8-10 and 9-10. These require also four additional points (0, 4, 7 and 8).

Remarks

The described assignment introduces some memory overhead compared to a single domain storage. Parts of the connectivity data will be stored on more than one domain. Furthermore the additional points increase the memory requirement for point variables and geometry data.

The relative amount of memory overhead depends on the number of points and edges on the domain interfaces compared to the total number of points.

The overhead in the example above is very large because the grid is so small. On grids with several thousands of points the relative overhead is much smaller even if the number of domains grows.

The computation of the edge cuts is performed by the grid partitioning algorithm. The resulting overhead depend on its quality.

6.3 Algorithmic description of the domain decomposition

The description is closely related to the datastructures used in the code. It should directly apply to any code using an *edge-based datastructure* and with some modification also to more general codes on unstructured grids.

First the description is restricted to the singlegrid case. The extension to multigrid is discussed later.

The algorithms are described in the sequence that corresponds to the data flow of the running program.

6.3.1 Required data

The whole grid in edge based form is required. This is given by the following data:

1. **nfaces, npoints** (integer): Grid dimensions, number of faces and number of points.
2. **fpoint[nfaces][2]** (integer): For each face the indices of the two points / control volumes which this face separates. These faces are associated to edges of the primary grid.
3. **fnormal[nfaces][3]** (double): For each face the face vector (area times normal vector pointing from first to second control volume).
4. **pcoord[npoints][3]** (double): Spatial coordinates of the points.
5. **pvolume[npoints]** (double): Volumes of the control volumes associated to the points.
6. **bdrypart**: A linked list of boundary parts **bp** containing the following data:
   (a) **bp.nfaces** (integer): Number of boundary faces of this treatment type.
   (b) **bp.treatment** (integer): Treatment type of this part of the boundary.
   (c) **bp.fpoint[bp.nfaces]** (integer): Indices of the points associated to the faces. Note that inside one boundary part a point is referenced at most once, but a boundary point can be referenced from multiple boundary parts.
   (d) **bp.npoint[bp.nfaces]** (integer): Indices of *near* points of the face points. For each boundary face this is a point connected to the corresponding face point by a face (edge) [1].
   (e) **bp.fnormal[bp.nfaces][3]** (double): For each boundary face the face vector (area times normal vector pointing out of the computational domain).

[1] One should note that there are two types of faces: First the faces that are associated to edges for the primary grid and second the boundary faces that describe the boundary of the computational domain.

All this data is computed from the primary grid in the first level of the pre-processing algorithm. Here it is simply assumed that the data is available.

6.3.2 Grid partitioning

The grid partitioning is an independent part that can be realized in very different ways without effecting the other algorithms. For grid partitioning sophisticated public domain software is available, which can easily be coupled with the existing sources. The coupling is performed for the public domain grid partitioning software package **metis**. A simple self-coded partitioning algorithm is implemented in addition (used by default). More information about the partitioning algorithms is given in the chapter **Grid partitioners**. Here we restrict to give the input/output specifications.

Input

The grid partitioning algorithm in principle can take the whole grid data available plus some weighting information (see below). Of coarse also the number of sub-domains **ndomains** that will be created has to be specified.

For each point of the grid a weight **pointweight[npoints]** (integer) is given, specifying the relative computational costs for each node. These are currently defined for a point **P** by the number of edges that end in **P**. The weights are solver dependent and will have to be changed if the main work of the solver is no longer dominated by the loops over the edges (faces).

Output

The only output of the grid partitioning algorithm is the domain number (integer) for each point **P**, specifying which domain owns **P**. These numbers are stored in the field **pointdomain[npoints]** (integer).

Interface for external grid partitioning

In order to allow the use of external grid partitioners writing 'partitioning input data' or reading 'partitioning output data' can be invoked by using the appropriate parameter settings. The interface files are ascii files. The formats are suitable for using **metis** in stand alone mode (there are also other public domain partitioners using the same format, e.g. **chaco**).

6.3.3 Sub-grid definition

Input

For the sub-grid definition the whole data available at this stage is required.

Output

ndomains subgrids **subgrid[ndomains]** (structures). Each subgrid **sg=subgrid[i]**, **i** from 0 to **ndomains**-1 contains the following data which is very similar to the original edge based data from the whole grid (see section 6.3.1).

1. **sg.nownpoints, sg.naddpoints, sg.nallpoints, sg.nfaces** (integer): Sub-grid dimensions, number of points owned by this domain, number of additional points, number of own points plus number of additional points, number of faces.

2. **sg.ncommdomains** (integer): Number of domains this domain needs to communicate with. This number is in any case less than **ndomains**.

3. **sg.globalidx[sg.nownpoints]** (integer): For each own point the index this point has in the original full grid. These indices are not required by the flow solver but by post-processing tools to gather the solver's output data.

4. **sg.addpoint_owner[sg.naddpoints]** (integer): Domain numbers of the additional points' owners.

5. **sg.addpoint_idx[sg.naddpoints]** (integer): Point indices on the owner's domain.

6. **sg.sendcount[ndomains]** (integer): Number of points that are needed by the other domains and must be sent at communication events.

7. **sg.recvcount[ndomains]** (integer): Number of (additional) points that this domain needs from the other domains and must be received at communication events.

8. **sg.commpartner[sg.ncommdomains]** (integer): Sequence which this domain uses when communicating with the others. This array contains the domain numbers. If between two domains information needs to be exchanged in both direction (this is the usual case!) then the domain with the lower number sends first.

9. **sg.fpoint[sg.nfaces][2]** (integer): See Part 6.3.1.

10. **sg.fnormal[sg.nfaces][3]** (double): See Part 6.3.1.

11. **sg.pcoord[sg.nallpoints][3]** (double): See Part 6.3.1.

12. **sg.pvolume[sg.nallpoints]** (double): See Part 6.3.1.

13. **sg.bdrypart**: Linked list of boundary parts like in Part 6.3.1.

sg.fpoint[sg.nfaces][2], **sg.fnormal[sg.nfaces]** are face data of all faces from the whole grid that have at least one point owned by the domain stored in **sg**.

sg.pcoord[sg.nallpoints][3], **sg.pvolume[sg.nallpoints]** are point data of the own and the additional points.

sg.bdrypart are those boundary faces from the whole grid that have a boundary face point which is owned by the domain stored in **sg**.

Computation of the output data

For the computation of the subgrid data we need two additional integer arrays **pointidx[npoints]**, **allpointidx[npoints]** and integer variables **i, j, k, kk, n, nn, n1, n2, nown, k1, k2, nmin**.

1. Init the **sendcount** and **recvcount** arrays:

```
for k=0,...,ndomains−1 do
    for kk=0,...,ndomains−1 do
        subgrid[k].sendcount[kk] := 0
        subgrid[k].recvcount[kk] := 0
    done
done
```

2. Count number of points owned by each subgrid:

```
for k=0,...,ndomains−1 do
    subgrid[k].nownpoints := 0
done
for i=0,...,npoints−1 do
    k := pointdomain[i]
    subgrid[k].nownpoints := subgrid[k].nownpoints+1
done
```

3. Allocate the arrays **globalidx** and reset the **nownpoints** variables:

```
for k=0,...,ndomains−1 do
    allocate(subgrid[k].globalidx, subgrid[k].nownpoints)
    subgrid[k].nownpoints := 0
done
```

4. Assign the global indices and recompute the **nownpoints** variables:

```
for i=0,...,npoints−1 do
    k := pointdomain[i]
    subgrid[k].globalidx[subgrid[k].nownpoints] := i
    allpointidx[i] := subgrid[k].nownpoints
    subgrid[k].nownpoints := subgrid[k].nownpoints+1
done
```

5. Now loop over the subgrids and compute the remaining data. This is a quite long loop containing some sub-loops. Most of the data is computed here.

```
for k=0,...,ndomains−1 do
```

   (a) Shortcuts:

```
nown := subgrid[k].nownpoints
```

   (b) Init local indices:

```
for i=0,...,npoints−1 do
    pointidx[i] := npoints
done
for i=0,...,nown−1 do
    pointidx[subgrid[k].globalidx[i]] := i
done
```

(c) Count faces, find and count additional points.
Additional points are required for several reasons. The first additional points are introduced due to faces that connect own points with points owned by other domains. More additional points may be introduced due to the boundary faces: There may be near points on other domains while the face point is on domain **k**.[2]
The faces counted for domain **k** are those with at least one point owned by domain **k**. For the additional points due to boundary faces we inspect the near points of boundary faces with a facepoint owned by domain **k**:

```
subgrid[k].naddpoints := 0
subgrid[k].nfaces := 0
for j=0,...,nfaces−1 do
    if pointidx[fpoint[j][0]] < nown or pointidx[fpoint[j][1]] < nown then
        subgrid[k].nfaces := subgrid[k].nfaces+1
        for i=0,1 do
            if pointidx[fpoint[j][i]] = npoints then
                pointidx[fpoint[j][i]] := nown+subgrid[k].naddpoints
                subgrid[k].naddpoints := subgrid[k].naddpoints+1
            endif
        done
    endif
done
for each bp of the original whole grid do
    for j=0,...,bp.nfaces−1 do
        if pointidx[bp.fpoint[j]] < nown
        and pointidx[bp.npoint[j]] = npoints then
            pointidx[bp.npoint[j]] := nown+subgrid[k].naddpoints
            subgrid[k].naddpoints := subgrid[k].naddpoints+1
        endif
    done
done
```

(d) Allocate remaining point and face variables:

```
subgrid[k].nallpoints := nown+subgrid[k].naddpoints
allocate(subgrid[k].addpoint_owner, subgrid[k].naddpoints)
allocate(subgrid[k].addpoint_idx, subgrid[k].naddpoints)
allocate(subgrid[k].fpoint, subgrid[k].nfaces)
allocate(subgrid[k].fnormal, subgrid[k].nfaces)
allocate(subgrid[k].pcoord, subgrid[k].nallpoints)
allocate(subgrid[k].pvolume, subgrid[k].nallpoints)
```

(e) Copy the pointdata and set the additional point's references. Furthermore the **sendcount** and **recvcount** are updated:

```
for i=0,...,npoints−1 do
    n := pointidx[i]
    if n < npoints then
        subgrid[k].pcoord[n] := pcoord[i]
        subgrid[k].pvolume[n] := pvolume[i]
        if n ≥ nown then
```

[2] More reasons for additional points are given by the multigrid related datastructures, see section 6.4.

```
            n := n−nown
            kk := pointdomain[i]
            subgrid[k].addpoint_owner[n] := kk
            subgrid[k].addpoint_idx[n] := allpointidx[i]
            subgrid[k].recvcount[kk] := subgrid[k].recvcount[kk]+1
            subgrid[kk].sendcount[k] := subgrid[kk].sendcount[k]+1
        endif
    endif
done
```

(f) Copy the face data:

```
subgrid[k].nfaces := 0
for j=0,...,nfaces−1 do
    if pointidx[fpoint[j][0]] < nown or pointidx[fpoint[j][1]] < nown then
        subgrid[k].fpoint[subgrid[k].nfaces][0] := pointidx[fpoint[j][0]]
        subgrid[k].fpoint[subgrid[k].nfaces][1] := pointidx[fpoint[j][1]]
        subgrid[k].fnormal[subgrid[k].nfaces] := fnormal[j]
        subgrid[k].nfaces := subgrid[k].nfaces+1
    endif
done
```

(g) Compute the boundary parts of the subgrid. After the computation the list of boundary parts has to be cleaned up in the sense that empty parts have to be removed from the list (simple implementation, not described here.).

for each **bp** of the original whole grid compute **sbp** of the subgrid; do

i. Treatment is the same:

```
sbp.treatment := bp.treatment
```

ii. Count the faces that are to be copied:

```
sbp.nfaces := 0
for j=0,...,bp.nfaces−1 do
    if pointidx[bp.fpoint[j]] < nown then
        sbp.nfaces := sbp.nfaces+1
    endif
done
```

iii. Allocate the **sbp** data:

```
allocate(sbp.fpoint, sbp.nfaces)
allocate(sbp.npoint, sbp.nfaces)
allocate(sbp.fnormal, sbp.nfaces)
```

iv. Copy the data:

```
sbp.nfaces := 0
for j=0,...,bp.nfaces−1 do
    if pointidx[bp.fpoint[j]] < nown then
        sbp.fpoint[sbp.nfaces] := pointidx[bp.fpoint[j]]
        sbp.npoint[sbp.nfaces] := pointidx[bp.npoint[j]]
        sbp.fnormal[sbp.nfaces] := bp.fnormal[j]
        sbp.nfaces := sbp.nfaces+1
    endif
done
```

done

done

This ends the long loop over the subgrids.

6. Count the domains each domain has to communicate with. A pair of communication is only counted once. We negate the **sendcount** and **recvcount** values in this part to flag those communication pairs that were already counted. Later they will be negated again.

n := 0
for **k**=0,...,**ndomains**−1 do
  subgrid[k].ncommdomains := 0
done
for **k**=0,...,**ndomains**−2 do
  for **kk**=**k**+1,...,**ndomains**−1 do
    if **subgrid[k].sendcount[kk]**+**subgrid[k].recvcount[kk]** > 0 then
      subgrid[k].ncommdomains := **subgrid[k].ncommdomains**+1
      subgrid[kk].ncommdomains := **subgrid[kk].ncommdomains**+1
      n := **n**+1
      subgrid[k].sendcount[kk] := −**subgrid[k].sendcount[kk]**
      subgrid[k].recvcount[kk] := −**subgrid[k].recvcount[kk]**
    endif
  done
done

7. Now allocate the communication sequence arrays and reset the counts:

for **k**=0,...,**ndomains**−1 do
  allocate(**subgrid[k].commpartner, subgrid[k].ncommdomains**)
  subgrid[k].ncommdomains := 0
done

8. Last define the communication sequences. The communication sequences are important from two different points of view. First, dead looks have to be prevented. Second, the full bandwidth of a parallel computer should be used to keep communication as cheap as possible.

for **i**=0,...,**n**−1 do

   (a) Init the count limit such that each communication pair will be lower counted:

   nmin := **ndomains** · **ndomains**

(b) Loop over the possible communication pairs and find that non-counted one with the lowest communication sequence length:

```
for k=0,...,ndomains−2 do
    for kk=k+1,...,ndomains−1 do
        if subgrid[k].sendcount[kk]+subgrid[k].recvcount[kk] < 0 then
            n1 := max(subgrid[k].ncommdomains, subgrid[kk].ncommdomains)
            n2 := min(subgrid[k].ncommdomains, subgrid[kk].ncommdomains)
            nn := ndomains · n1+n2
            if nn < nmin then
                k1 := k
                k2 := kk
                nmin := nn
            endif
        endif
    done
done
```

(c) Set the communication (**k1**, **k2**):

```
subgrid[k1].sendcount[k2] := −subgrid[k1].sendcount[k2]
subgrid[k1].recvcount[k2] := −subgrid[k1].recvcount[k2]
subgrid[k1].commpartner[subgrid[k1].ncommdomains] := k2
subgrid[k2].commpartner[subgrid[k2].ncommdomains] := k1
subgrid[k1].ncommdomains := subgrid[k1].ncommdomains+1
subgrid[k2].ncommdomains := subgrid[k2].ncommdomains+1
```

done

6.4 Extension to the multigrid case

6.4.1 Additional grid and sub-grid data

For the connection between the gridlevels three datafields are added to the grid as well as to the sub-grid container[3]. Because we have more than one whole grid in the multigrid case (fine grid and coarse grids) we use containers for the whole grids too. There is one grid-container for each gridlevel.

Assume, we have two consecutive gridlevels (a fine grid **fgrid** and a coarse grid **cgrid**). On the fine grid we store for each point the index of the corresponding point on the coarse grid (the coarse grid controlvolume into which the fine grid controlvolume is fused). On the whole grid this is stored in the array
fgrid.parent_pnt[fgrid.npoints] (integer)
On the coarse grid we store for each point the indices of the child-points on the fine grid (the fine grid controlvolumes that were fused into the coarse grid controlvolume). There are variable counts of fine grid controlvolumes fused into the coarse grid controlvolumes so the indices are stored into two arrays:
cgrid.child_pnt[fgrid.npoints] (integer),
cgrid.child_pntidx[cgrid.npoints+1] (integer).

[3]The number three is not true for the finest and the coarsest gridlevel. On the the finest gridlevel the **child_pnt** and **child_pntidx** fields are not required while on the coarsest grid the **parent_pnt** field is not required.

The first array contains the fine grid point indices while the second contains indices into the first to locate the child point indices for specific coarse grid points. That means the fine grid point indices for a specific coarse grid point **cp** are stored in
cgrid.child_pnt[cgrid.child_pntidx[cp]]
to **cgrid.child_pnt[cgrid.child_pntidx[cp**+1**]**−1**]**.

On the subgrids the variables are named identically. Assume now that we have a subgrid **fsg** from **fgrid** and a subgrid **csg** from **cgrid**, both with the same domain index. The fields are here **fsg.parent_pnt[fsg.nownpoints]** (integer) and
csg.child_pnt[???] (integer),
csg.child_pntidx[csg.nownpoints+1**]** (integer).
The childpoints from the own points on **csg** are the same as the childpoints of these points on the whole coarse grid **cgrid**. This defines the dimension of **csg.child_pnt** as the sum of the own point's children count.

6.4.2 More *additional points* on the subgrids

The **parent_pnt** and **child_pnt** indices define some connectivity between points (on different gridlevels). This may require additional points on the subgrids. If again **fsg** is a subgrid of the fine grid **fgrid** and **csg** is a subgrid of the coarse grid **cgrid**, both with the same domain index (on the same processor), then we require that all coarse grid points referenced by **fsg.parent_pnt[]** are available on **csg** and all fine grid points referenced by **csg.child_pnt[]** are available on **fsg**. This extends the part where the additional points are defined and counted (see section 6.3.3, item 5c on page 140).

6.5 Communication interface

The domain decomposition described so far can or should be contained in an external pre-processing program. For the flow-solver itself also a small additional pre-processing step has to be added to obtain some index fields.

Assume now, we have the data for the **k**-th subgrid **sg** = **subgrid[k]** available (see Part 6.3.3).

At several stages of a flow computation different pointdata values of the additional points are required. Therefore a communication interface is constructed.

6.5.1 Communication index fields

In principle the references to the additional points are stored in the fields **sg.addpoint_owner** and **sg.addpoint_idx**. For efficiency reasons the message passing between two domains has to be performed in one block. To achieve this, additional index fields are stored. Furthermore other domains also need data that is owned by domain **k**. The indices of these points are currently only known by the other domains. From **sg.sendcount** only the counts of these points are known.

The indices are stored in the following two (two-dimensional) arrays:
sendindex[ndomains][] (integer)
recvindex[ndomains][] (integer)
The second dimensions depend from the first index **kk** and are given by **sg.sendcount[kk]** and **sg.recvcount[kk]**.

The field **recvindex** can be computed from the fields **sg.addpoint_owner** and **sg.addpoint_idx**:

```
for i=0,...,sg.ncommdomains do
    kk := sg.commpartner[i]
    n := 0
    for j=0,...,sg.naddpoints−1 do
        if sg.addpoint_owner[j]=kk then
            recvindex[kk][n] := sg.nownpoints+j
            n := n+1
        endif
    done
done
```

The field **sendindex** can not be computed locally but from the other domains. The message passing facility is already used for this task. On domain **k** the already computed index field **recvindex** is used to pass the **sendindex** field of the communication partners to those. We therefor use a communication buffer (see below):
commbuffer[] (variable type)
This buffer is completely hidden by the message passing interface and is used as container for different kinds of data.

The index fields are exchanged as follows with the communication partners:

```
for i=0,...,sg.ncommdomains do
    kk := sg.commpartner[i]
    if kk > k then (first send)
        for j=0,...,sg.recvcount[kk]−1 do
            commbuffer[j] := sg.addpoint_idx[recvindex[kk][j]-sg.nownpoints]
        done
        senddata(commbuffer, sg.recvcount[kk], kk)
        recvdata(commbuffer, sg.sendcount[kk], kk)
        for j=0,...,sg.sendcount[kk]−1 do
            sendindex[kk][j] := commbuffer[j]
        done
    else (first receive)
        recvdata(commbuffer, sg.sendcount[kk], kk)
        for j=0,...,sg.sendcount[kk]−1 do
            sendindex[kk][j] := commbuffer[j]
        done
        for j=0,...,sg.recvcount[kk]−1 do
            commbuffer[j] := sg.addpoint_idx[recvindex[kk][j]-sg.nownpoints]
        done
        senddata(commbuffer, sg.recvcount[kk], kk)
    endif
done
```

The sending and receiving protocol that is used here is the same that will be used below for sending and receiving the pointdata.

Remarks

It is assumed here that functions senddata() and recvdata() for sending data to another task and for receiving data from another task are available. These functions are taken from a message passing library like *MPI*.

The communication buffer **commbuffer** that is shared for all sends and receives is allocated with a sufficient size earlier.

6.5.2 Communication routine

For the regular communication calls of the solver only one routine is provided. It is named exchange_pointdata(). It performs the whole data exchange for one (two-dimensional) point-data array. The two-dimensional array is passed as an one-dimensional array and the index computation is done internally.

For this routine the **sendindex** and **recvindex** arrays are required (see above).

Communication is required if updated values of the additional points are needed for a calculation. Each domain has to send parts of its own data to other domains using the **sendindex** information and to receive the additional data using the **recvindex** information.

Algorithm for blocking sends and receives

The used communication protocol ensures that no dead-locks can happen. Furthermore, if the number of domains is large then a large number of communication calls can be executed independently from each other in parallel, utilising the ability of independent communication calls on a powerful parallel computer.

The protocol uses the **sg.commpartner** field and the rule that if with one communication partner a send and a receive is required, then the domain with the lower index first sends and then receives while the domain with the higher index first receives and then sends.

The algorithm then looks as follows:

The parameters are the dimension **dim2** and a pointdata array:
pointdata[dim2 * sg.nallpoints] (double). The domain index of **sg** is **k**.

```
for kk=0,...,sg.ncommdomains do
    n := sg.commpartner[kk]
    if n > k then (first send)
        for i=0,...,sg.sendcount[n]−1 do
            for j=0,...,dim2−1 do
                n1 := i·dim2+j
                n2 := sendindex[n][i]·dim2+j
                commbuffer[n1] := pointdata[n2]
            done
        done
        senddata(commbuffer, dim2·sg.sendcount[n], n)
        recvdata(commbuffer, dim2·sg.recvcount[n], n)
        for i=0,...,sg.recvcount[n]−1 do
            for j=0,...,dim2−1 do
                n1 := i·dim2+j
                n2 := recvindex[n][i]·dim2+j
                pointdata[n2] := commbuffer[n1]
            done
        done
    else (first receive)
        recvdata(commbuffer, dim2·sg.recvcount[n], n)
```

```
        for i=0,...,sg.recvcount[n]−1 do
            for j=0,...,dim2−1 do
                n1 := i·dim2+j
                n2 := recvindex[n][i]·dim2+j
                pointdata[n2] := commbuffer[n1]
            done
        done
        for i=0,...,sg.sendcount[n]−1 do
            for j=0,...,dim2−1 do
                n1 := i·dim2+j
                n2 := sendindex[n][i]·dim2+j
                commbuffer[n1] := pointdata[n2]
            done
        done
        senddata(commbuffer, dim2·sg.sendcount[n], n)
    endif
done
```

Algorithm for non-blocking sends and receives

If the message passing interface supports non-blocking sends and receives (*MPI* does!) then a simpler algorithm can be used. Note that the use of non-blocking sends and receives usually reduces the communication time. So this version should be preferred[4].

To use this alternative an own communication buffer for each send and receive call is required. Assume we have an array of sendbuffers **sndbuf[sg.ncommdomains][]** and of receivebuffers **rcvbuf[sg.ncommdomains][]**, all of sufficient size.

The prototype of this routine is exactly the same as for the blocking case. The parameters are the dimension **dim2** and a pointdata array:
pointdata[dim2 * sg.nallpoints] (double). The domain index of **sg** is **k**.

First all receives are started. Then the data to be sent is copied into the send buffers and the sends are started. After this all domains have to wait until the sends and receives are finished. Finally the received data is copied from the communication buffers to the additional points' memory:

```
for kk=0,...,sg.ncommdomains do
    n := sg.commpartner[kk]
    start_recvdata(rcvbuf[kk], dim2·sg.recvcount[n], n)
done
for kk=0,...,sg.ncommdomains do
    n := sg.commpartner[kk]
    for i=0,...,sg.sendcount[n]−1 do
        for j=0,...,dim2−1 do
            n1 := i·dim2+j
            n2 := sendindex[n][i]·dim2+j
            sndbuf[kk][n1] := pointdata[n2]
        done
    done
```

[4] When compiling the code it is currently a compile time option to choose either the blocking or non-blocking routine

```
    start_senddata(sndbuf[kk], dim2·sg.sendcount[n], n)
done
wait_sends_recvs_finished()
for kk=0,...,sg.ncommdomains do
    n := sg.commpartner[kk]
    for i=0,...,sg.recvcount[n]−1 do
        for j=0,...,dim2−1 do
            n1 := i·dim2+j
            n2 := recvindex[n][i]·dim2+j
            pointdata[n2] := rcvbuf[kk][n1]
        done
    done
done
```

6.6 Modifications of the flow solver

The algorithms described so far are modifications that should not effect the further development of the flow solver.

Of course, also the flow solver itself has to be modified. The whole concept is designed to keep the amount of changes low and to make them as simple as possible.

6.6.1 Loops over all points or only over the own points?

One part of modifications is introduced by the two different number of point dimension. One has to decide if a loop over the points of a domain has to include all points or only the own points.

6.6.2 Adding exchange_pointdata() calls

For efficiency reasons the use of communication calls using exchange_pointdata() is not such simple like it could be in principle. In principle after each loop that computed some values for the own points but not correctly for all points a call to exchange_pointdata() should be done.

Loops over the faces but also loops over the boundary faces usually compute the data for the own points but not completely for the additional points because not all faces connecting the additional points are given on a domain.

Often it is possible to postpone a communication call to save computation time or even avoid some communication calls. Especially the possibility to avoid a communication call should be considered very carefully because communication usually is the main reason for pure parallel performance.

A typical example where communication calls can be avoided is the flux computation of the flow solver. It starts with the computation of the convective fluxes which first needs a loop over the edges and second over the boundary faces. In the viscous case another loop over the edges has to be performed to compute the viscous fluxes. These three loops all add values to the **flux** variables of the points. There is only one communication call required after all these sums are computed.

6.6.3 Using global reduction operations

For the current flow solver reduction operations are not essentially required, but it is a nice feature of the solver that it prints some integral numbers (residual, lift, drag and others) after each multigrid cycle.

To compute these numbers correctly more or less effort has to be spent. Powerful communication interfaces like *MPI* provide global reduction operations like sums. For residual, lift and drag only sums are required. For the residual one has to compute the sum over all domains before computing the square root of the sum. For drag and lift the boundary integrals and also the area (length) of the body have to be computed as global sums before the integrals are divided by the area.

6.7 Grid partitioners

The grid partitioners currently implemented are a simple self-coded partitioner and the public domain software package **metis**. The simple partitioner can not achieve the partitioning quality of sophisticated methods, such that more communication overhead decrease the parallel performance. When using only a small number of processors (lets say 2 or 4) the effect remains negligible. In such a case it is of advantage that the partitioning itself is much faster. Going to massive parallel computations the use of other partitioners is recommended.

6.7.1 Simple partitioner

The default partitioner computes the edgecuts according to coordinates. If two partitions have to be computed it is compared if the partitioning at **x** = const (**x** is the position on half the way between **x-max** and **x-min**) requires less cuts of edges than a cut at **y** = const or **z** = const. The best of the three cuts is used. If three domains have to be computed the partitioning is performed by dividing first in 2 subdomains weighted with 1/3 and 2/3. The second subdomain is then divided again in 2 partitions with equal weights. All other numbers of subdomains are computed using the same algorithm recursively, e.g 7 subdomains are obtained by dividing first in 2 domains weighted with 3/7 and 4/7. The first is then partitioned in 3, the second is two times divided in 2 partitions.

6.7.2 Metis partitioner

The **metis** partitioner is a public domain software package. Sources or further information can be obtained from the web page `http://www.cs.umn.edu/~karypis`. The sources are coupled with the code without performing modifications (except in the Makefiles) such that a substitution of the current version by a later update is as simple as possible. Only the old sources have to be replaced and the object lists in the Makefiles have to be adapted. Metis has been selected for the coupling to the code by **IBK** within the FASTFLO project. A more detailed documentation of **metis** has been prepared by **IBK** and can be found in the annex.

7 Unsteady

7.1 Deforming mesh without GCL

The pseudo temporal change of the conservative variables $\vec{W}$ for the physical time τ can be derived for a deforming mesh as:

$$\frac{\partial}{\partial t}\vec{W} = -\frac{1}{V}\cdot\int\int_{\partial V}\bar{\bar{F}}\cdot\vec{n}dS - \frac{\partial}{\partial\tau}\vec{W} - \vec{W}\cdot\frac{\partial V}{\partial\tau} \tag{7.1}$$

The time derivative of the conservative variables $\frac{\partial}{\partial\tau}\vec{W}$ and of the control volumes $\frac{\partial V}{\partial\tau}$ are computed by a finite difference in physical time τ.

7.2 Deforming mesh with GCL

With the geometric conservation law GCL,

$$\frac{\partial V}{\partial\tau} = \int\int_{\partial V}\vec{U}\cdot\vec{n}dS \qquad \vec{U} = \frac{\partial\vec{X}}{\partial\tau} \tag{7.2}$$

the pseudo temporal change of the conservative variables $\vec{W}$ for the physical time τ can be derived as:

$$\frac{\partial}{\partial t}\vec{W} = -\frac{1}{V}\cdot\int\int_{\partial V}\bar{\bar{F}}\cdot\vec{n}dS - \frac{\partial}{\partial\tau}\vec{W} - \vec{W}\cdot\int\int_{\partial V}\vec{U}\cdot\vec{n}dS \tag{7.3}$$

Where the time derivative of the conservative variables $\frac{\partial}{\partial\tau}\vec{W}$ and of the grid coordinates $\vec{U}$ are computed by a finite difference in physical time τ. The Integral $\int\int_{\partial V}\vec{U}\cdot\vec{n}dS$ is only evaluated once for every physical time step.

8 Grid-Adaptation Indicator

8.1 Introduction

Automatic grid adaptation is an essential task for a fully automatic CFD system. It is composed of two parts: a strategy how to refine the given grid and the detection of the locations and the strengths of the refinement. One classical approach for adaptation indicators is the use of gradients or undivided differences of any suitable flow variable. This approach can detect flow phenomena, when taking for instance the primitive variables into account (e.g. shocks are detected because of large gradients) and is followed by the gradient-based indicator described below. Alternatively one can reconstruct the flow values on the edge midpoints from both sides, just like it is done within the solver, and then take the difference of the two obtained values. This leads to an indication of the curvature, but it might as well be interpreted as the gradient-based indicator added to the difference of gradients. We call this method reconstruction-based. Another approach is to formulate directly an indication for the local discretization error. This strategy is followed by the residual-based indicator described below.

Each of these indicators can be applied depending on the parameter settings. We decided to provide each type of indicator, because it is not clear up to now which strategy will be of advantage in future applications. The fact that the residual-based indicator can be interpreted as a real measure for the discretization error seems to be of advantage on the one hand. On the other hand it provides no directional information as the gradient-based and the reconstruction-based indicator do (which is the gradient in the direction of an edge).

8.1.1 Gradient-Based Indicator

A classical approach for adaptation indicators is the use of gradients or differences of any suitable flow variable. The approximated gradient $G_{(V)}$ of a variable V in discrete form is $\Delta V/h$, with $\Delta V = V_{p_1} - V_{p_2}$ i.e. the difference between the point values of the two points p_1 and p_2 connected by one edge, where h is the length of the edge. We write the gradient-based indicator as:

$$I \quad = \quad \Delta V h^{\alpha} \tag{8.1}$$

A widely used formulation is $\alpha = 1$, i.e.: $G_{(V)}h^2$. The advantage of scaling the indicator with a positive value of α is that the adaptation stops automatically in the corresponding area after several cycles also when discontinuities are present in the flow field. Our three-dimensional numerical tests have shown that values of $\alpha < 1$ may be of advantage in order to strengthen the grid refinement near surfaces where the cell sizes are typically some orders less than in the farfield region. We use $\alpha = 0.5$ as default value. Our choice of ΔV is:

$$\Delta V = MAX\left(C_{\phi_i}\frac{\Delta\phi_i}{(\Delta\phi_i)_{ref}}\right) \tag{8.2}$$

with $0 <= i < N$ and N being the number of different variables considered. The current implementation is:

$$\phi_i = \{\rho, |\vec{v}|, P_t, H_t\} \tag{8.3}$$

P_t is the total pressure, H_t is the total enthalpy. $|\vec{v}|$ is defined as

$$|\vec{v}| = \sqrt{(u_{(p_1)} - u_{(p_2)})^2 + (v_{(p_1)} - v_{(p_2)})^2 + (w_{(p_1)} - w_{(p_2)})^2}$$

The weights C_{ϕ_i} are parameters enabling to choose different combinations of the single parts of the indicator (to be set to zero in order to turn off ϕ_i).

The reference values $(\Delta\phi_i)_{ref}$ are for an equilibrated scaling of each part of the indicator with

$$(\Delta\phi_i)_{ref} = MAX((\Delta\phi_i)_j), \text{ (for all grid points j).}$$

The above described indicator refines the grid in the whole flowfield with equal weights. Because it can be very expensive to refine e.g. the whole shock region with the same weight as the leading edge, we introduced surface weights. They allow to strengthen the refinement near the surface such that e.g the pressure loss around a leading edge decreases without obtaining too small cells farer away.

The modification reads as:

$$I = \Delta V h^{\alpha}(1 + C_w w) \tag{8.4}$$

with $w = 0$ for all inner edges, $w = 1$ for edges having one surface point and $w = 2$ for edges having two surface points. Good results have been obtained when setting the parameter C_w to $C_w = 1$. With $C_w = 0$ (default) the surface weights are disabled.

8.1.2 Residual-Based Indicator

A residual based indicator developed at the DLR has been successfully used for two dimensional computations. Thus it has been extended to the following three-dimensional formulation. We write the steady state Euler equations for compressible flows in the form:

$$\sum_{i=1}^{3} \partial_{x_i} f_i(u) = 0$$

with u being the vector of the conserved variables. Then we define the residual r_h by inserting the solution vector u_h, which is an approximation of the exact solution u, into the above equation:

$$r_h = \sum_{i=1}^{3} \partial_{x_i} f_i(u_h)$$

It can be shown (see [65]) that under smoothness assumptions the local error on a cell Ω can be bounded from below and above by $||hr_h||_{L_2(\Omega)}$. Although, the smoothness assumption is violated when discontinuities are present, $||hr_h||_{L_2(\Omega)}$ behaves well, as two-dimensional test cases have shown. From an engineering point of view it is quite useless to know that the error of the approximative solution will tend to zero if the mesh size tends to zero (due to the scaling with h) because of limited computer resources. Thus, a change of the scaling factor h can be considered to h^{α} and $0 \leq \alpha \leq 1$. In three-dimensional numerical tests it turned out that $\alpha = 0$

is the best choice. The computation of the residual is performed on the primary grid cells. It would not make sense to compute it on the auxiliary grid cells because the solver is constructed to converge the residual on these cells to zero. Linear fluxes are assumed over the primary cells Ω. Thus r_h is constant over Ω and the residual based indicator I is computed as:

$$\begin{aligned} I(\Omega) &= ||r_h||_{L_2(\Omega)} \\ &= \sqrt{|\Omega|}\sqrt{\sum_{i=0}^{n}(r_h)_i^2} \end{aligned} \tag{8.5}$$

where $n = 5$ is the number of equations and $|\Omega|$ is the volume of Ω. The values of the flux functions at the 4 points of a tetrahedra are computed from the given u_h of the flow variables. Thus the remaining step for the computation of r_h is to compute the partial derivatives of the flux functions resulting in a linear system of equations. For the computation of the residuals on prisms the same procedure is used by subdividing first each prism into three tetrahedra.

The indicator values are related to the volume elements. All values of elements connected by a single grid point are averaged and stored as point value. The sum of the indicator values of two points connected by an edge builds the indicator edge value used by the edge-based adaptation algorithm.

8.1.3 Reconstruction-Based Indicator

Since the solver is second order accurate with the aid of gradients, we want to calculate the differences of reconstructed values. Therefore we need the spatial gradient on each point for each flow value. For their calculation see below.

Let the ϕ_i be the different flow values, then the indicator on the edge e with the points p_1, p_2 is defined as:

$$I(e) = \max_i |(\phi_i(x_{p_1}) + \frac{1}{2}x_e \cdot grad(\phi_i(x_{p_1}))) - (\phi_i(x_{p_2}) - \frac{1}{2}x_e \cdot grad(\phi_i(x_{p_2})))| \cdot |x_e|^\alpha,$$

where $x_e = x_{p2} - x_{p1}$ and α is a userdefined exponent.

8.1.4 Calculation of Gradients on the Primary Grid

The information about the dual cells and their volumes or faces is not available within the adaptation module, therefore we cannot use the *Green-Gauss* formula as in the solver. We use instead a *least-square* approach, where we only need distances between neighboring points. For a function $\phi : \Re^3 \mapsto \Re$ we calculate the gradient at the point x_0 with the set of neighboring points $\{x_1, ..., x_{N_p}\}$ by:

$$grad(\phi(x_0)) = \sum_{j=1}^{N_p} M_0^{-1} w_{j,0}(\phi(x_j) - \phi(x_0))(x_j - x_0), \tag{8.6}$$

where $w_{j,0} := \frac{1}{\|x_j - x_0\|^2}$ and the weighting matrix is defined by

$$M_0 := \sum_{j=1}^{N_p} w_{j,0}(x_j - x_0)^T(x_j - x_0). \tag{8.7}$$

8.1.5 Calculation of y_+ Values

With the aid of the gradient of the velocity vector V we can calculate the vorticity:

$$\omega(x) = \sqrt{(\partial_2 V_1(x) - \partial_1 V_2(x))^2 + (\partial_3 V_2(x) - \partial_2 V_3(x))^2 + (\partial_1 V_3(x) - \partial_3 V_1(x))^2} \tag{8.8}$$

Now let x_W be a point on a solid wall, n the corresponding surface normal and x_N the near point of x_W. Then y_+ is defined by:

$$y_+ := \sqrt{\rho(x_W)\mu(x_W)\omega(x_W)} \cdot \frac{n(x_W - x_N)}{|n|} \tag{8.9}$$

8.1.6 Application of Adaptation Indicators for Local Grid Refinement

Applying the adaptation indicators for local grid refinement an additional parameter and its influence has to be discussed. For the local grid refinement it has to be decided which edges have to be bisected. It is natural to select the edges with large indicator values. In order to refine a certain number of edges a threshold has to be chosen (which is in the current implementation a selection of the percentage of enlarging the grid dimensions). Different indicators can lead to the same grid refinement for one threshold and to a different refinement for an other threshold. This becomes obvious when considering the example that the distribution of two indicators is very similar except for local maxima occurring on different positions. A high threshold leads to the selection of some maximum values at different grid positions. For a low threshold much more edges are selected, which may lead to an identical refinement (the limiting case is that all edges will be bisected). This tendency can be observed employing the above described indicators. Enlarging a grid in a single adaptation step of more than the double size can lead to similar results. Enlarging a grid over several steps only by, lets say, 40% per step the different indicator characteristics become visible. Considering the example of a transonic flow past a wing the residual-based indicator leads to a strong refinement of the leading edge region and a weak refinement of the shock region. Using the gradient-based indicator a stronger shock refinement is obtained. The reconstruction-based indicator is somewhere inbetween.

9 Grid-Adaptation Algorithm

9.1 Introduction

The aim of this algorithm is to build, given a mesh and a solution on it, a new conforming mesh as a result of certain element refinements and in case of a viscous calculation on a hybrid mesh as a result of redistribution of points on wall normal rays depending on the given solution and the corresponding refinement-indicator. For the refinement of the different types of elements, the primary grid is composed of, it is also referred to e.g. [48].

The grid refinement is performed by calculating the list of edges of the mesh and then marking edges to be bisected depending on the refinement-indicator strategy. In the next step by several loops over the elements additional edges are marked considering only allowable refinement cases (for details see below) in order to get a conforming mesh.

Afterwards the point-coordinates and the interpolated solution values on the new points are calculated and the new elements are constructed, thereby storing the 'critical'-refinement cases, such that in a next adaptation step these 'critical'-elements can be treated separately. These 'critical'-elements are new elements that come from non-isotropic refinement cases. The reason why these elements have to be treated separately is that otherwise there might occur new elements with very small angles.

On hybrid grids the wall normal pointdistribution can be changed, such that a userdefined y_+ value is reached all over the solid wall and the intersection of prismatic and tetrahedral (or prismatic and hexahedral in quasi two-dimensional grids) elements can be improved in regard of the corresponding dual grid.

All the used files, apart from the parameter file and the selected-elements file are in the *NetCDF*-format. Therefore the implementation is influenced by the capabilities and restrictions of this library.

9.2 Basic Ideas

The following points are the basic ideas of what is done in the tool. A detailed description of the different modules is given in the implementation part of this description later on.

1. There are some informations about the grid and its edges we need. First of all this is the list of edges containing the two point numbers. With this we get element-edge lists of the elements, containing the edges that form the element.

2. Now we have to judge with a refinement-indicator and a strategy, which edges are to be bisected. If the mesh has been adapted before we have to consider the 'critical'-elements and treat them by different methods, which means that there are edges which are not allowed to be bisected.

3. Several loops over the prismatic elements are made in this step. If a prism has marked edges mark other edges of the prism such that the corresponding refinement is allowable and store the number of marked edges in each prism. This has to be done until no further edges are marked in a whole loop.

4. Step 3. is done with all tetrahedral elements.

5. Since step 4. might have caused new marked edges on the boundary shared by tetrahedra and prisms, steps 3 and 4 have to be repeated until no new additional edges are marked.

6. After step 5. the marking of edges is conforming in the sense that the following corresponding construction of the elements leads to a conforming mesh. However, before we can start this construction, we have to run one loop over all elements, that is prisms, tetrahedra, triangles and quadrilaterals and count the number of new elements in order to allocate the arrays needed.

7. By a loop over the edges the new point-coordinates and the interpolated solution values on the new points are calculated and the new point-numbers are stored in an edge-flag. If interpolation of the new surface points or redistribution of points in the boundary layer is required, wall normal rays have to be extracted and the translation is prolongated over these rays. With the same datastructures the fixing of y_+ is performed.

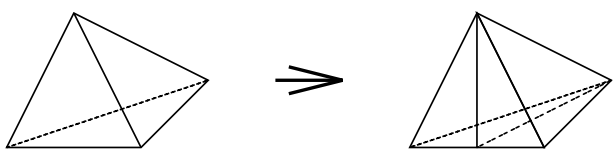
8. This is the last step, where the new elements are constructed and stored in arrays. If 'critical'-refinement cases emerge the involved element- and point-numbers are stored in the new grid-file.

9.2.1 Allowable Refinement Cases

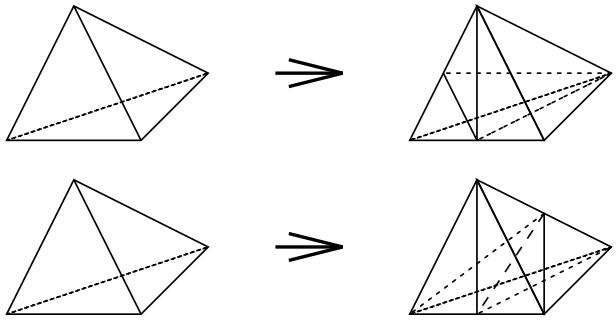
Since we do not want the adapted mesh to get much worse, according to the quality of elements, than the initial mesh, we only allow certain refinement cases on elements.

We start by describing the refinement-cases on tetrahedra since these are the only real $3D$-elements so far supported. The refinement cases for prisms, triangles, hexahedra and quadrilaterals are then just projections of the tetrahedra refinement-cases.

- There are ten different cases of tetrahedra refinement on initial meshes (that means, meshes that have not been adapted before and therefore determine the quality of all the following resulting meshes).
  - critical-refinement
    - $(1:2.1)$-refinement: One edge of the tetrahedron is marked and the old tetrahedron is split into two new ones.

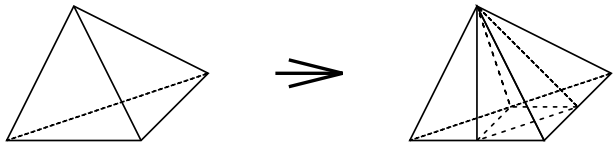


    - $(1:3.2)$-refinement: Two edges are marked and these edges are connected by one point. The old tetrahedron is split into three new ones.

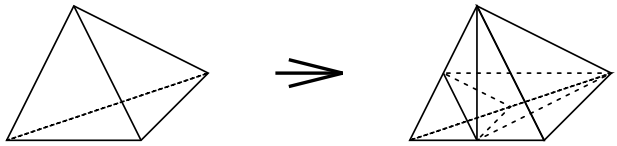


* $(1:4.2)$-refinement: Two edges are marked and these edges are not connected. The old tetrahedron is split into four new ones.

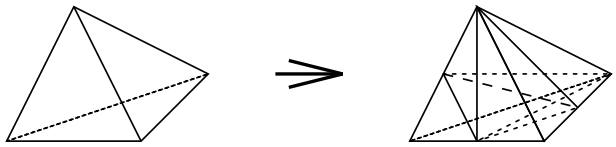
* $(1:4.3\,face)$-refinement: Three edges are marked and these edges build a triangle. The old tetrahedron is split into four new ones.



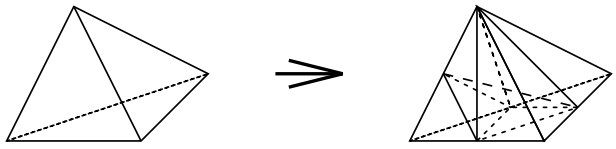
* $(1:4.3)$-refinement: Three edges are marked and these edges are connected by one point. The old tetrahedron is split into four new ones.



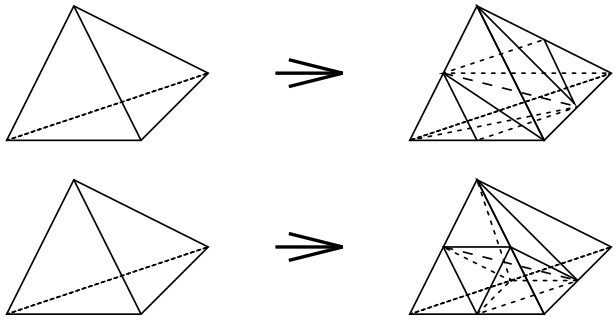
* $(1:5.3)$-refinement: Three edges are marked and these edges are neither connected nor adjacent. The old tetrahedron is split into five new ones.



* $(1:6.4\,face)$-refinement: Four edges are marked and three of these edges build a triangle. The old tetrahedron is split into six new ones.



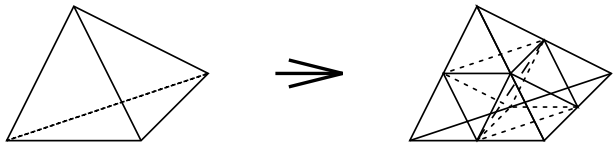
* $(1:6.4)$-refinement: Four edges are marked and these are pairwise opposite. The old tetrahedron is split into six new ones.



* $(1:7.5)$-refinement: Five edges are marked. The old tetrahedron is split into seven new ones.

– isotropic-refinement

* $(1:8.6)$-refinement: All edges (that is six) are marked. The old tetrahedron is split into eight new ones.



Since this splitting is not unique, we have to set a rule how to construct the new tetrahedra. 24 new edges are determined by the midpoints of the old edges. For the 25-th edge (that is the inner edge) the choice is free. We get 'best' new tetrahedra by choosing the smallest inner diagonal of the old tetrahedron and then constructing the new ones along this line.

- Consequently there are three different cases of triangle and two cases of prism refinement on initial meshes. (We only describe the refinement on triangles because refinement on prisms just means the same procedure done on both triangles of the prism.)

  – critical-refinement

    * $(1:2)$-refinement: One edge is marked and we get two triangles out of one.
    * $(1:3)$-refinement: Two edges are marked and we get three triangles out of one.

  – isotropic-refinement

    * $(1:4)$-refinement: All edges (that is three) are marked and we get four triangles out of one.

- On quadrilaterals the situation is straightforward, because either they are refined or they are not.

- On hexahedra the situation is the same as on quadrilaterals.

On meshes that have been adapted before there is a list of elements that are obtained from 'critical'-refinement cases. These are all critical cases for tetrahedra, $(1:2)$ and $(1:3)$ for triangles and $(1:2)$ for prisms.

If an element is in the list of 'critical'-elements and has marked edges, there are the following steps to refine it. (Note that there are edges which are not allowed to be marked at all.)

- On all elements:
  1. We reconstruct the 'parent'-element with its old edge-connectivity (that is the element this one comes from).
  2. All edges in the 'parent'-element that are needed to construct all current 'child'-elements are marked.
  3. Now the edges in all current 'child'-elements that connect two points in the corresponding 'parent'-element are checked for being bisected in all current 'child'-elements and the same edges are marked in the 'parent'-element, such that we get allowable refinement cases.
  4. If other edges are bisected in the 'child'-elements as described above, we have to perform further refinements and we must mark all edges in the 'parent'-element and therefore the corresponding edges in the 'child'-elements. (Note that the only possible refinement case is an isotropic one for the parent).
  5. We construct the new elements depending on the refinement case of the 'parent'-element.
- Further refinement on critical elements:
  1. A subgrid is build of the new isotropic refined elements with the same edge-connectivity as an initial mesh would have.
  2. We search in all old 'child'-elements for bisected edges and store the information in the subgrid, too (only if the edges are the same !).
  3. At last we treat this subgrid as if it was an initial grid and then we can construct all possible refinement cases in this further refinement.

9.2.2 The List of 'Critical' Elements

After constructing a non-isotropic element ('critical') we store the following data:

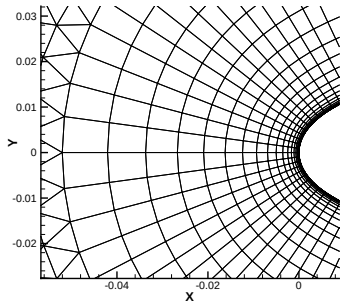
- The refinement case of the element.
- Four (tetrahedron), six (prism) or three (triangle) point-numbers of the 'parent'-element.
- An offset number, which points to the beginning of the 'child'-elements (e.g. a tetrahedron has been refined $(1:3.2)$ and the new tetrahedra are stored at position 201, 202 and 203, than the offset is set to 201).

Starting an adaptation on an previously adapted mesh, this information is read and stored in suitable arrays, as the new refinement depends on this information.

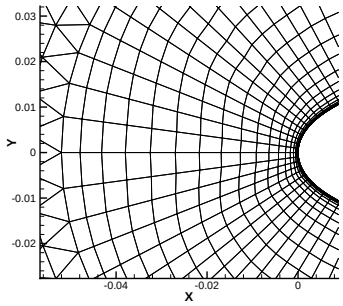
9.2.3 Dealing with Semi-Structured Parts

Adaptation on hybrid grids, which are usually used for viscous calculations, does need some further considerations. Simply bisecting edges in the semi-structured parts (prisms in three-dimensional grids and hexahedra in two-dimensional grids) cannot be desirable. If we want to conserve the element types, which we do in the boundary layer for instance, an edge in wall normal direction that is bisected would imply a whole new layer of elements. Instead of adding

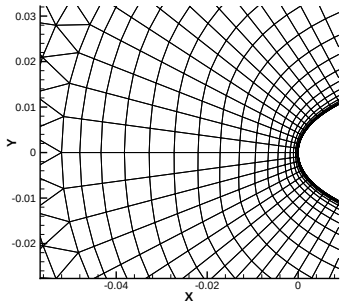
points in this direction we move the available points depending on the y_+ value of the first field point and the average length of the surrounding edges of the last point. That means we determine the first and the last point distance on each wall normal ray and then redistribute the points according to the solution of the resulting one-dimensional boundary value problem. The new values are then projected to the quadratic spline of the initial distribution. In order to achieve a certain smoothness of the grid, the first and last distances are smoothed by averaging of the values of the neighbors.



initial grid
$y_+ \approx 1.5$
no relation



adapted grid
$y_+ \approx 0.5$
last distance
about 50% of average



smoothed adapt. grid
$y_+ \approx 0.5$
last distance
about 50% of average

9.3 Implementation

The following steps have to be done. The order corresponds to the implementation and is chosen in order to save memory. Each point is explained in detail below:

- read the parameters
- read the grid and the point-coordinates
- orientate surface normals on surface elements and calculate the missing grid information
- read the curve information (only on meshes with defined bar_2 elements)
- read the critical-elements lists (only on adapted meshes)
- indicate edges which are not allowed to be bisected (only on adapted meshes)
- calculate the value of the chosen indicator on each edge
- mark edges to be bisected depending on the refinement-indicator strategy
- mark additional edges until the corresponding grid is conforming
- count the number of new elements and elements involved in critical-refinement cases
- calculate the new point-coordinates and -numbers
- distribute the translation of new surface points along the piles (on hybrid grids only)
- adapt the wall normal point distribution to the wanted values (on hybrid grids only)
- update curves (only on meshes with defined bar_2 elements)

- interpolate the solution onto the new points and write the new restart solution for the adapted grid
- write the new curve information (only on meshes with bar_2 elements)
- construct the new tetrahedra thereby writing the tetrahedra blockwise; count and write the critical tetrahedra blockwise
- construct the new prisms; count and write the critical prisms blockwise
- construct the new hexahedra
- construct the new triangles and quadrilaterals; count and write the critical triangles blockwise
- write the point-coordinates; write all remaining elements and surfmarkers where necessary

9.3.1 Reading the Grid and the Point-Coordinates

The primary grid file is read and the following structure is built:

```
typedef struct
{
  int ntetra, nprism, nhexa, nstri, nsquad;
  int (*tetrapnt)[4];
  int (*prismpnt)[6];
  int (*hexapnt)[8];
  int (*stripnt)[3];
  int (*squadpnt)[4];
  int *surfmarker;
} VolumeGridConnectivity;
```

The point-coordinates are stored in:

```
double *xc[3];
```

(Note that the case $nhexa > 0$ is currently only supported in $2D$ grids.)

9.3.2 Calculating the Missing Grid Information

The edge based grid is stored in the following structure:

```
typedef struct
{
  int nedges;
  int **edge_pnt;

  int **tetra_edge;
  int **prism_edge;
  int **hexa_edge;
```

```
  int **tri_edge;
  int **quad_edge;

  int **cri_tetra;
  int **cri_prism;
  int **cri_tri;
} EdgeBasedGrid;
```

For the calculation of these arrays the point to element connectivity is needed. Since the numbers of elements surrounding a point can differ in an unstructured mesh and we do not want to waste memory, this information is stored in two one-dimensional arrays. One of them containing the indexes and the other one containing the actual element numbers.

Example for the calculation of the point surfaceelement connectivity:

After these connectivity lists are available (the above has to be done for volumeelements as well) we can compute the array ($**edge\_pnt*$) of all edges of the grid, containing the two point-numbers and two flags. $edge\_pnt[edge][0]$ indicates whether the edge has to be bisected or not (set to 0), $edge\_pnt[edge][1..2]$ are the point-numbers and $edge\_pnt[edge][3]$ indicates whether the edge is allowed to be bisected or not (set to 0).

Then we need the element-edge connectivity, containing the numbers of the edges of the element and a flag.

(Note that we do not store all edges of hexahedra, prisms and quadrilaterals, but only the ones that form a triangle in a prism or the ones that are not in the predefined offset direction in a hexahedra, respectively.) In these $\_edge[elem][0]$ is a symbolic constant that indicates the current refinement case in dependence to the number of marked edges (set to 0). And $\_edge[elem][1..x]$ are the numbers of the edges that form the element (e. g. a tetrahedron has 6 edges and a triangle 3).

In prisms the edges are ordered. That means $prism\_edge[edge][1]$ connects point 0 and 1, $prism\_edge[2]$ connects point 1 and 2, $prism\_edge[3]$ connects point 2 and 0, $prism\_edge[4]$ connects point 3 and 4 and so on.

In hexahedra we only need four edges and a normal vector to determine which square lies in the plane (Note that refinements on hexahedra are supported only in $2D$-meshes).

9.3.3 Reading the Curve Information

If there is a list of pairs of pointnumbers, which indicates curve edges, in the grid-file, this list is stored in the following structure:

```
typedef struct
{
   int nbar_2;
   int (*barpnt)[2];
   int ncurves;
   int *curveidx;
} CurveEdge;
```

ncurves defines the number of different surface curves. *curve_idx* stores the start and end index of each curve (if $ncurves > 1$).

9.3.4 Reading the Critical Elements Lists

If the grid has been adapted before and there are elements coming from critical-refinement cases, there are the mentioned lists in the grid-file. In order to store these we allocate the different arrays of pointers, which are in the edge based datastructure.

```
int *cri_tetra[ntetra];
int *cri_prism[nprism];
int *cri_tri[nstri];
```

Now if an element is in the list of criticals, the corresponding pointer has to point to an allocated array such that we can put the needed information in it. As mentioned above the size of these arrays depends on the element type.

The first entry of these arrays is always the refinement case of the 'parent'- element (e.g. REF_1_2 for $(1:2.1)$-refinement and REF_1_3 for $(1:3.2)$-refinement). The next ones are the point-numbers of the 'parent'-element and the following one is the offset number of the 'child'-elements (this means that all elements coming from a critical refinement are stored in one block).

- $int[5]$ for triangles
- $int[6]$ for tetrahedra
- $int[8]$ for prisms

9.3.5 Indicating Edges Which are not Allowed to be Bisected

If there are critical elements in the grid certain edges are not allowed to be bisected, because if one of these elements has to be refined these edges will vanish.

In critical triangles and prisms there are one or two such edges, respectively. In critical tetrahedra there are more than one edge that is not allowed to be bisected and these edges are always depending on the refinement case of the 'parent'-element.

Here is an example for a $(2:4)$-refinement on a triangle. The dotted edge is the one that vanishes after the refinement.

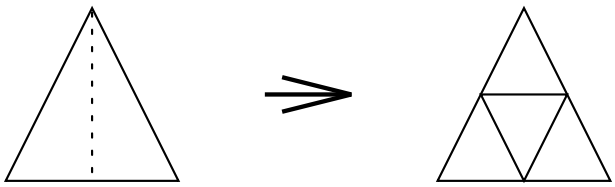


Figure 9.1: $(2:4)$ refinement case

We indicate these edges by setting $edge\_pnt[3]$ to 1.

9.3.6 Calculating the Values of the Chosen Indicator on Each Edge

Since we now have access to the list of edges, we can apply the indicator function to the two points of each edge and store the results in an array:

```
double edge_ind_vals[nedges];
```

9.3.7 Marking Edges to be Bisected

If a selected-elements filename is given in the input-file, which is an ASCII-file containing the number of elements and their element-numbers which have to be refined, the element-numbers are read and all old edges of the element (that means edges that existed in the previous grid if there was one) are marked.

Otherwise, after the calculation of the indicator value for each edge, the average of the minimum and the maximum of these values is taken as thresholds and each edge with an indicator value above this average is marked. Afterwards the additional edges are computed (see below) and the resulting number of new points is compared to the userdefined number of new points. If it is to small or high the threshold is decreased or increased respectively.

9.3.8 Marking Additional Edges

Since after the previous step the bisection of edges will usually not lead to a conforming mesh, we have to bisect additional edges. This has to be done first of all for the prisms, afterwards for the tetrahedra and then be repeated until no additional edge marking occurs in a whole loop.

Marking Additional Edges in Prisms

In prisms the marking of additional edges is straightforward. As we have sorted the edges in prisms, we can run over the prisms and mark the edges on the opposite triangular side of marked edges in each prism. Only in the case that two edges (in a prism triangle) are marked, we have to mark the third one (and the opposite one of course). We then store a symbolic flag that indicates the refinement case (that is NO_REF, REF_1_2 or REF_1_4) of the prism.

If the grid has been adapted before there might be critical prisms and they have to be treated separately in the following way:

If one or more of the edges of a critical prism is marked we build a temporary prism and reconstruct the parents edge connectivity (and so the old refinement case). Then we mark all edges which have been in the parent-element and still exist in the 'child'-elements (the points of the parent element are in the critical-list). Furthermore we have to mark the same edges in the temporary prism that represents the 'parent'-element with its current refinement status and check this prism for an allowable refinement case (Note, it is possible to mark some additional edges to get an allowable refinement case). If edges are marked that have not been in the parent element, we have to mark all edges in the temporary prism and we have to set the flag to $REF\_2\_4\_PLUS$ in all 'child'-elements to indicate that further refinement is necessary on the elements.

We run loops over the prisms until no prism-flag changes. Since there is a finite number of edges only, this procedure will terminate and moreover the resulting prism refinement will be conforming.

Marking Additional Edges in Tetrahedra

In tetrahedra the same procedure is more sophisticated, because it means real $3D$-refinement and therefore there are more allowable refinement cases. We run loops over all tetrahedra and check the number of marked edges and whether they are connected by each other or not.

- One edge is marked and we have a $(1:2.1)$-refinement so the flag is set to REF_1_2.
- Two edges are marked so we have to look whether they are connected $(1:3.2)$ or not $(1:4.2)$ and set the flag to REF_1_3 or REF_1_2TWO.
- Three edges are marked and three different cases are possible. The edges form a triangle $(1:4.3face)$ or are all connected by one point $(1:4.3)$ or form a chain $(1:5.3)$. The flags to be set are REF_1_4FACE, REF_1_4, REF_1_5.
- Four edges are marked and three of them form a triangle $(1:6.4face)$ or their are pairwise opposite $(1:6.4)$. Here the flags are REF_1_6FACE or REF_1_6.
- Five edges are marked and we get a $(1:7.5)$-refinement with the flag REF_1_7 to be set.
- All edges are marked and we get a $(1:8.6)$-refinement with the flag REF_1_8.

With each refinement status we have the number of marked edges, so if the number of marked edges is changed, we must change the flag depending on the new refinement case.

If the grid has been adapted before there might be critical tetrahedra and they have to be treated separately in the following way:

As described above we reconstruct for each tetrahedron the 'parent'-element with the old edge-connectivity (this means we store the new points that are calculated in the adapted mesh before in the same edges they are coming from) so we can construct the critical-elements immediately (the flag - $tetra\_edge[elem][0]$ - is set to the first value in the critical_list - $cri\_tetra[elem][0]$). After this we have to check if edges in critical-elements ('child'-elements) are marked that have not been in the 'parent'-element. If one of those edges is marked we have to mark all edges in the 'parent'-element (and therefore all 'old' edges in the critical-/'child'-elements) and set the flag to REF_1_8PLUS which means that further refinement is necessary on the new elements. If only edges are marked that have been in the 'parent'-element we can easily determine what refinement case is necessary to get a conforming mesh, by marking these edges in the 'parent'-element, too. At last we have to check the current refinement status of the 'parent'-element.

We run loops over the tetrahedra until no tetrahedron-flag changes. For the same reason as for the prisms this procedure will terminate and the resulting tetrahedra refinement will be conforming.

Here is an example for three critical-tetrahedra obtained by a $(1:3.2)$-refinement and one 'old' edge is marked. The 'parent'-element is reconstructed and the edge informations are stored in the 'parent'-element. Then a $(1:4.3)$-refinement is performed on the 'parent'-element. Thereby the three dotted edges vanish due to the refinement.

9.3.9 Counting the Number of New Elements

In order to allocate the memory needed and to define the variables of the new grid file we have to know how many elements of each type will be in the new grid. We also must know how many 'critical'-elements of each type we will have to write.

In order to get these values we run a loop over each element type (except the tetrahedra) and by looking at the flag we find out how many new elements there will be. For the surface-elements (triangles and quadrilaterals) we have to count the number of marked edges and store the refinement case in the flag (which depends on the number of marked edges), in order to calculate the needed values as described above.

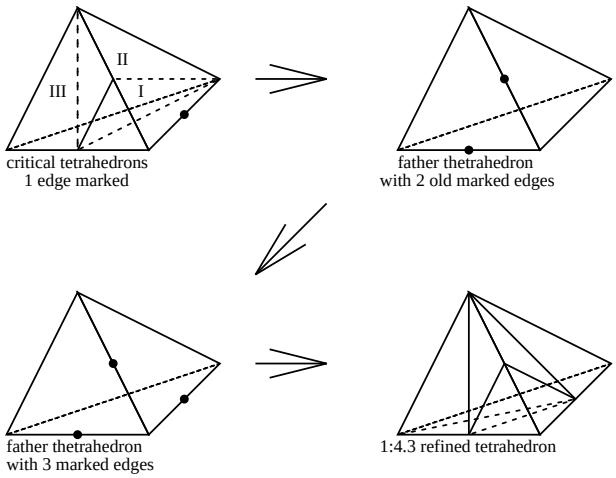


Figure 9.2: $(3.2:4.3)$ refinement case

Note that in the all further-refinement cases for tetrahedra we do not know at that time how many new tetrahedra will be constructed, because this depends on along which diagonal we refine. Therefore we cannot calculate the number of new tetrahedra elements and criticals (This is done in the construction routines).

9.3.10 Calculating the New Points

Now we count the number of marked edges and allocate the memory for the new point-coordinates, since a marked edge means a new point. So far quasi-quadradic spline interpolation of the new points on surfaces is supported. While calculating the new point-coordinates we store the new point-number (always added to the number of old points) in $edge\_pnt[0]$.

Interpolation of New Points on Surfaces

The interpolation of a new point between two old points is based on Bézier-spline interpolation and therefore requires the two old point-coordinates and the normals on the old points depending on the surrounding surface-elements.

The surface normal on a point p is calculated through

$$n_p = \frac{1}{\| \sum_{i=1}^{N_p} \frac{\alpha_{e_i}}{|a_{e_i}+b_{e_i}|} n_{e_i} \|} \cdot \sum_{i=1}^{N_p} \frac{\alpha_{e_i}}{|a_{e_i}+b_{e_i}|} n_{e_i},$$

where N_p is the number of neighboring surface-elements e_i of p, α_{e_i} is the angle at p of e_i, n_{e_i} is the unit normal on e_i and a_{e_i}, b_{e_i} are the edge-vectors of e_i from p.

If there is no information about curve edges available, while these sums are calculated we look at the angle between two normals on elements with a common edge and count how often it is greater than our maximum-surface-angle. If there is a curve edge list, we run a loop over these curve edges and count how often a point appears. If more than one curve is defined, we expect each curve to have unique start- and endpoints. After that we set this number to 1 if it was 0 and to 3 if it was 1.

In both cases we interpret this number as the number of surface patches at point p. This means in particular, if this number differs from 1, we do not have a unique normal on point p.

This number is stored for each surface point in the array `int no_norms[snpoints];` where *snpoints* denotes the number of surface points. The normals are only stored for points with one unique normal in `double *dxc[3];`

After this we run a loop over all edges thereby looking at the two points $p1$ and $p2$ of marked ones. We now interpret the values $no\_norms[p1]$ and $no\_norms[p2]$ in the following way:

1. $no\_norms[p1] = no\_norms[p2] = 1$:

   both points are on the same smooth surface patch

2. ($no\_norms[p1] > 1$ and $no\_norms[p2] = 1$) or
   ($no\_norms[p1] = 1$ and $no\_norms[p2] > 1$) :

   both points are on the same smooth surface patch but one point is also a curve point

3. $no\_norms[p1] = no\_norms[p2] = 2$:

   both points are on the same smooth curve

4. ($no\_norms[p1] > 2$ and $no\_norms[p2] = 2$) or
   ($no\_norms[p1] = 2$ and $no\_norms[p2] > 2$) :

   both points are on the same smooth curve but one point is an end point of this curve

5. $no\_norms[p1] > 2$ and $no\_norms[p2] > 2$:

   the edge between these two points builds a whole curve

In all cases apart from 1 we do not have a normal on the involved point. The calculation of course depends on the case.

In case 3 and 4 the two points are interpreted as curve points. So in order to calculate their normals, we look at the neighboring points and if there is one unique neighboring point (otherwise linear interpolation is performed), which is not the other edge point, we are able to calculate a normal on the point with two neighbor points. In order to do so we restrict the case to the plane through these three points and then calculate the normal in the usual two-dimensional way.

In case 2 and 4 only one normal is given, so that we have to set an appropriate boundary condition in order to calculate the other one. We chose constant curvature as the boundary condition. That is, we reflect the given normal at the plane through the midpoint of the two points with the vector from one point to the other as the normal.

In case 5 no normal information is available, so that we perform a linear interpolation in this case.

Now either we have performed the interpolation (linear) already, or we do have two normals for each edge. We now calculate the intersection of the two planes on these normals fixed at the corresponding point. If the intersection differs from a line (which means the normals were colinear), we perform linear interpolation again. Otherwise we project the midpoint of the regarded edge onto the line. We now have three points for each edge. These three points are now the control points of our Bézier-spline. The new point is now the intersection of this function with the plane through the midpoint of the edge with the vector from one edge point to another as normal.

However, the translation of a new linear point on the surface to the splined one can damage the grid. If we think of stretched tetrahedra or prisms with high aspect ratios, this damage is very likely. Therefore we check if the translation shifts the new point outside adjacent elements. In case these are tetrahedra, we set the point coordinate back to the linear one. In case of prisms

we translate all the new points in the corresponding pile and check the resulting elements. If these are damaged, we take the linear point.

There are two values to be set. One is *MAX_ANGLE* which is the angle between two normals from which on linear interpolation is performed. The other one is *EPS* which should be the accuracy of the surface points in the given mesh.

9.3.11 Adapting the Wall Normal Pointdistribution

In order to redistribute points along a wall normal ray, we need these rays in an ordered sense. We decided to store these rays edgewise. That means, for each edge lying on a solid wall we have an ordered list of the corresponding edges in the same ray. We store these like e.g. the point to element connectivity. So we have an array with the indexes

```
int layer_listidx[nedges + 1];
```

and a two-dimensional array with the actual data

```
int (*layer_list)[3];
```

where the first entity is the number of the edge and the second and fourth are the pointnumbers of the adjacent points (this holds only for three-dimensional grids, where each edge in a prismatic layer is connected to four prisms and therefore has two unique neighbor points). However, since we want to extract lines, the edgenumbers are signed. A positive number means that the corresponding points are on the same line as the underlying point and vice versa. The computation of this information is done by looping over the prisms. First of all we flag all edges on the wall and then we run loops over the prisms flagging the unflagged edges with the bottom edgenumber and the actual layer number (wall points have layernumber 0. These loops are repeated until no edge is flagged. After that the filling of the layer list is straightforward.

As soon as we have computed these arrays, we have access to each ray. Now we can calculate y_+ on the first field point and the desired distance on the last point. After this is done, these two values for each ray are limited and smoothed. With the resulting values we redistribute the points and project them onto the curve spline. The same method is applied to each solution variable in order to save computational time for the restart process.

9.3.12 Interpolating the Solution

The interpolation of the solution values at the new points works as for the point-coordinates. So far only linear interpolation is supported. All conserved quantities given in the file are interpolated.

9.3.13 Constructing and Writing the Tetrahedra and the Critical Tetrahedra

Now we run a loop over each element-type looking at the flag in the element-edge array and construct the resulting elements.

- all elements:

  - flag *REF_1_2* (*refine_tetra_1*)
  - flag *REF_1_3* (*refine_tetra_2*)
  - flag *REF_1_4TWO* (*refine_tetra_4_TWO*)
  - flag *REF_1_4FACE* (*refine_tetra_4_FACE*)
  - flag *REF_1_4* (*refine_tetra_4*)
  - flag *REF_1_5* (*refine_tetra_5*)
  - flag *REF_1_6FACE* (*refine_tetra_6_FACE*)
  - flag *REF_1_6* (*refine_tetra_6*)
  - flag *REF_1_7* (*refine_tetra_7*)
  - flag *REF_1_8* (*refine_tetra_8*)
    In this case we have to find the shortest inner diagonal and then refine along this line.

- critical elements (reconstruction):
  - all flags (*construct_tmp_tetra*)
  - flag *REF_1_2* (*restore_old_1_2_connect*)
  - flag *REF_1_3* (*restore_old_1_3_connect*)
  - flag *REF_1_4TWO* (*restore_old_1_4_TWO_connect*)
  - flag *REF_1_4FACE* (*restore_old_1_4_FACE_connect*)
  - flag *REF_1_4* (*restore_old_1_4_connect*)
  - flag *REF_1_5* (*restore_old_1_5_connect*)
  - flag *REF_1_6FACE* (*restore_old_1_6_FACE_connect*)
  - flag *REF_1_6* (*restore_old_1_6_connect*)
  - flag *REF_1_7* (*restore_old_1_7_connect*)

- further refinements on critical elements:
  - flag *REF_1_8_PLUS*
    In this case we have to perform first an isotropic refinement (*refine_tetra_8*), then we construct a temporary grid of new elements (*build_sub_grid*), search all the edges that were not in the 'parent'-element and copy the information into the temporary grid (*search_plus_cases*) and at last check and construct all tetrahedra as described above (*construct_all_plus_cases*).

Since we did not know the number of tetrahedral elements before their construction, we define the corresponding *netcdf*-variable to *NC_UNLIMITED* and write each tetrahedron after its construction to a queue (and then blockwise to the grid-file).
For the same reason we cannot determine the exact number of critical tetrahedra, so we build a linear list and store the needed information (case, points and offset) for each critical-refinement-case. After the construction of the tetrahedra we count all critical-elements, define the corresponding *netcdf*-variable and write the list blockwise to the critical-grid-file.

9.3.14 Constructing and Writing the Remaining New and Critical Elements

We run a loop over each element-type looking at the flag in the element-edge array and construct the resulting elements. If critical cases occur or critical-elements from a previous adaptation are not changed the involved critical information has to be written to the grid file. Then these new elements have to be stored in the *VolumeGridConnectivity*-array considering the element numbers.

- The construction routines for prismatic elements:
  - non-critical elements:
    - flag *REF_1_2* (*refine_prism_2*)
    - flag *REF_1_4* (*refine_prism_4*)
  - critical elements:
    - flag *REF_2_4* (*refine_prism_2_4*)
  - further refinements on critical elements:
    - flag *REF_2_4_PLUS* (*refine_prism_2_p*)
- The construction/reconstruction routines for triangular elements:
  - all elements:
    - flag *REF_1_2* (*refine_tri_2*)
    - flag *REF_1_3* (*refine_tri_3*)
    - flag *REF_1_4* (*refine_tri_4*)
  - critical elements (reconstruction):
    - all flags (*construct_tmp_tri*)
    - flag *REF_1_2* (*restore_tri_from_1_2*)
    - flag *REF_1_3* (*restore_tri_from_1_3*)
  - further refinements on critical elements:
    - flag *REF_1_4_PLUS*
      In this case we construct the elements in the same way as described in the tetrahedra section (*refine_tri_4*), (*build_sub_tri_grid*), (*search_plus_cases*) and (*construct_plus_cases*).
- The construction routines for hexahedral elements:
  - There only is one case for these elements, as they either are split into two new ones or not. Consequently there are no critical cases.
    - flag *REF_1_2* (*refine_hexa_1_2*)
- The construction routines for quadrilateral elements:
  - There only is one case for these elements, as they either are split into two new ones or they are not. Consequently there are no critical cases.
    - flag *REF_1_2* (*refine_quad*)

In any critical case (that means a new one or one that still exists from a previous adaptation) the involved elements and the parent-element points are stored blockwise and written to the grid-file.

9.3.15 Writing the Elements, Curve Edges and Point-Coordinates

Since after the construction of the tetrahedra the point-coordinates are not needed anymore we write them before constructing and writing the remaining element-types.

For surface-elements we also have to consider the surface-marker, that means the child-elements have to have the same marker as their parent-element had.

10 Periodic Boundaries and Actuator Disks in the Primary Grid of TAU

10.1 Introduction

The actuator disk in the TAU-Code represents the classical model of momentum theory of a zero thickness surface which increases the momentum and the energy of the flow. The application of an external force via a zero thickness surface leads to jump discontinuities in the state of the flow. The fluxes are conservative through the disk. The discretization should maintain these properties, a sharp rendering of jump discontinuities and a conservative balancing of fluxes.

10.2 Grid Topology

To render jump discontinuities the grid needs two points at the same geometric position. Because zero edges between a point and its copy do not exist these points define the surface of an inner periodic boundary with the identity as transformation. Figure 10.2 shows the topology of the grid. The points on the periodic and the shadow half are geometrical identical but topological different. At the boundary of the disk the periodic halves are closed by a fix point line similar to the rotation axis of a rotary periodic boundary. Periodic pairs of fix point lines are geometrical and topological identical. Figure 10.4 shows the dual grid cells of a cut section through the disk. Due to the periodicity two boundary cells complete each other to an inner cell with two points. This enables a conservative balancing of fluxes through the disk and a sharp rendering of jump discontinuities. Actuator disks intersecting with walls double the points at the intersection line. Free boundaries are closed by fix point lines. Figure 10.3 shows the grid topology of an actuator disk intersecting with a wall and one with a free inner boundary as it is used for helicopter rotors.

10.3 CAD Model

Before the grid generation starts boundary groups have to be defined for the surface panels of the CAD model. If an actuator disk would be constructed as a watertight domain the unique identification of the periodic and the shadow half would not be possible because they are geometrical identical. To get a unique identification the actuator disk has to be constructed as single panel for which an orientation has to be specified. With the CAD interface of the grid generator a special actuator group has to be defined which doubles the surface during the grid generation. By convention the doubling takes place to the opposite of the orientation. As a preprequisite the grid generator has to accept a CAD model with a non watertight domain. Figure 10.1 shows the CAD model of an actuator disk with oriented surface.

10.4 NetCDF Format for Periodic Boundaries

Actuator disks represent a new type of periodic boundary with the identity as transformation. Information is stored in the same format as for rotatory or translatory periodic boundaries. The format of periodic information is described at the example of a small grid partitioned into two domains with a translatory and rotatory periodic boundary. Figure 10.7 shows the domains of the grid. The overlapping zone is one element layer. The points are printed with the local point number. The global point number of the non partitioned grid has been printed as subscript. The points which are not part of the overlapping zone are own points of the domain and are marked red. The points of the overlapping zone are denotiated as add points and are marked magenta. For add points there is a distinction if they are connected to own points by edges or elements. Add points which are connected by elements and not by edges are denotiated external points. But for the periodic format this is not of importance. The total number of points in the grid, the sum of nown plus nadd points, is denotiated as nall points. When periodic boundaries are partitioned periodic pairs may be subdivided so that a periodic point is part of the domain but its periodic partner does not belong to the domain. Points which are connected to the domain only by periodicity and not by overlapping are denotiated as per points and are marked orange. For these points there exist parallel extensions similar to the extensions for points in the overlapping zone. The NetCDF format of periodic information can store multiple periodic boundaries of translatory, rotatory or actuator disk type and has the following form:

```
dimensions:
    two;
    eight;
    no_of_perbdryidx;
    no_of_perbdry ;
    no_of_perpairidx ;
    no_of_perpair ;
    no_of_perpoints ;

variables:
    int perbdryidx(no_of_perbdryidx) ;
    double perinfo(no_of_perbdry, eight) ;
    int permarker(no_of_perbdry, two) ;
    int perpairidx(no_of_perpairidx) ;
    int perpair(no_of_perpair, two) ;
    int perpoint_owner(no_of_perpoints);
    int perpoint_id(no_of_perpoints);
```

The lists and the dependencies between the lists are shown in figure 10.8 and figure 10.9 for both domains of the grid in in figure 10.7. Periodic boundaries are stored in series ordered according to their periodic type. The order of periodic types is fixed to translation, rotation and actuator disk. The index list perbdryidx stores the start index of the periodic boundaries of each periodic type. The dimension of an index list is one more than the number of start indices to calculate the number of elements for each start index by subtracting it from its successor. So the last entry contains a virtual start index of a not existing following periodic type. The list permarker contains the marker pairs of all periodic boundaries ordered periodic - shadow. The list perinfo stores geometric information of each periodic boundary as tuples of eight doubles. The entries depend of the periodic type and are shown in figure 10.6. Translatoric periodic boundaries store the translation vector oriented periodic - shadow. Rotatory periodic boundaries store the

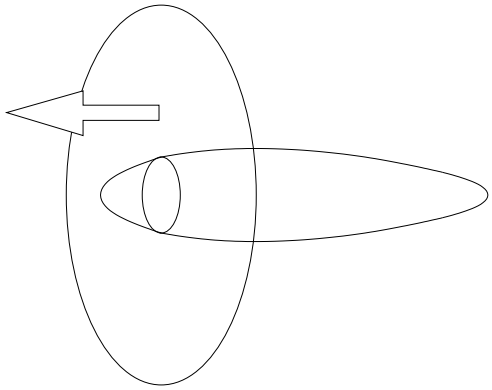


Figure 10.1: CAD model with single oriented panel.

unit axis of rotation, oriented mathematical positive for the rotation periodic - shadow, an axis point and the number of periodic sections. The number of sections as integral value is preferred instead of the periodic angle to avoid round off errors. Actuator disks store the unit axis of rotation by doubling oriented inside to the grid, the center and the outer and inner radius of the disk. The index list perpairidx stores the start index of the periodic pairs for each periodic boundary. The list perpair stores the points ids of the periodic pairs ordered periodic - shadow. Fix points are stored as periodic pair with identical point id.

For partitioned periodic pairs there exist the additional lists perpoint_owner and perpoint_id. These lists refer to points which are connected by periodicity but not by overlapping to points on the domain. These points get virtual point ids in the list perpairs which count up from nall. The offset in the list perpoint_owner and perpoint_id is calculated by subtracting nall from the point id. The list perpoint_owner stores the owner domain in which the point is own point and the list perpoint_id the local point id in the owner domain. These lists correspond to the lists addpoint_owner and addpoint_id for points of the overlapping zone. As shown in figure 10.5 actuator disks do not need parallel extensions because of the geometrical identity periodic pairs can always be shifted to the same domain.

For the TAU-Code the list perpairs includes points of the overlapping zone. The lists perbdryidx, perinfo and permarker are global. They store information of all periodic boundaries of the non partitioned grid. The list perpairidx has the dimension of all periodic boundaries but refers to the periodic pairs locally existing on the domain. If a periodic boundary in the grid does not exist on the domain the difference of the start index and its successor is zero.

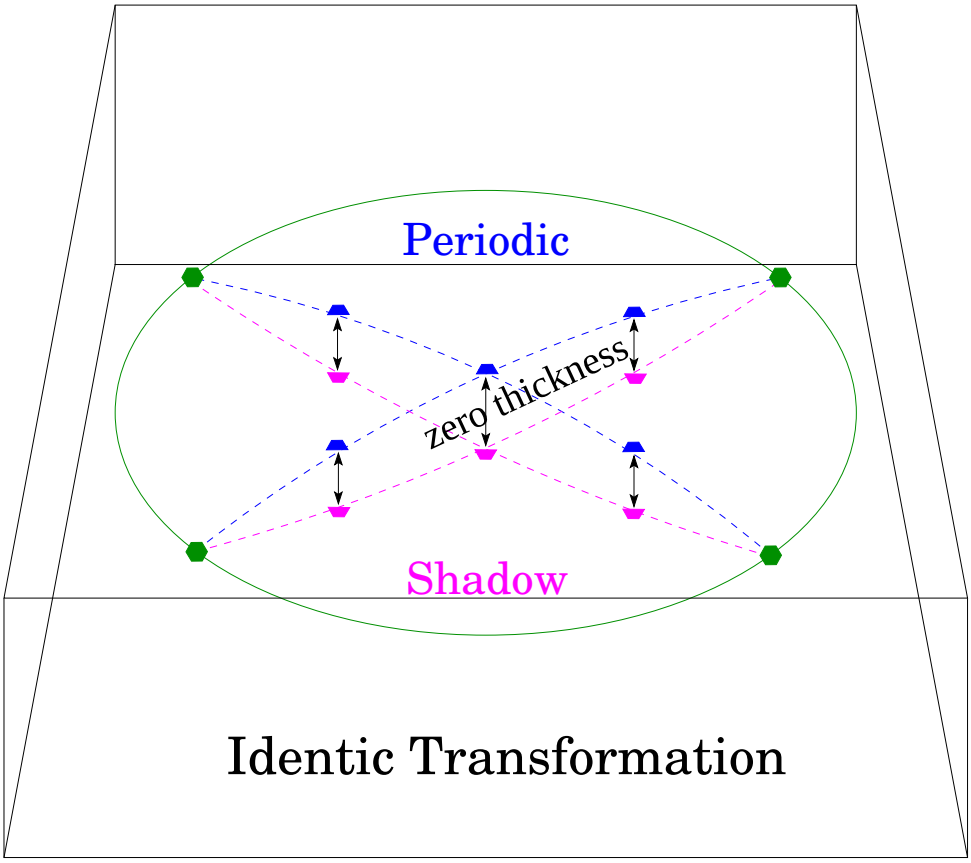


Figure 10.2: Grid topology of actuator disk periodic boundaries.

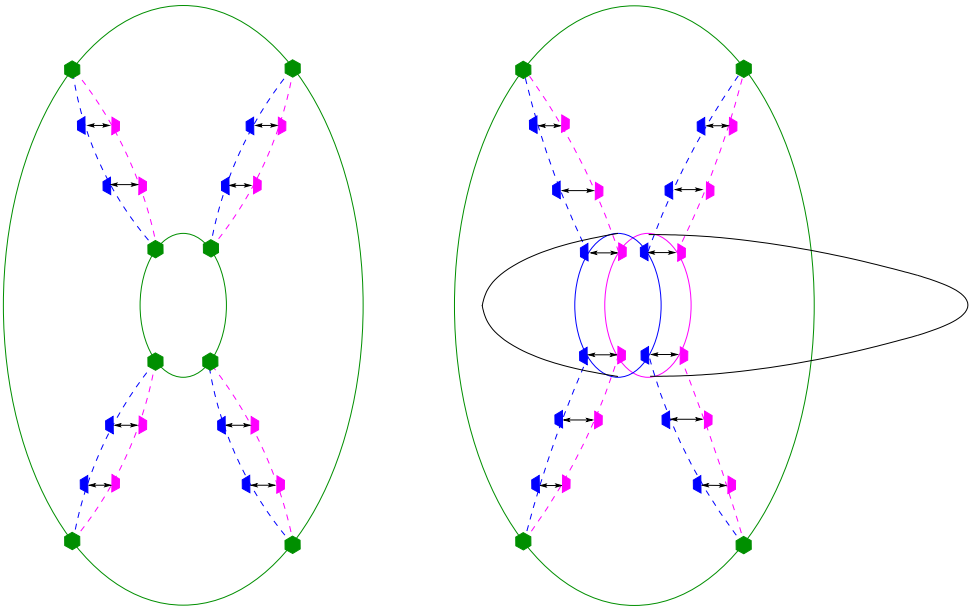


Figure 10.3: Actuator disks with inner hole and intersecting with wall.

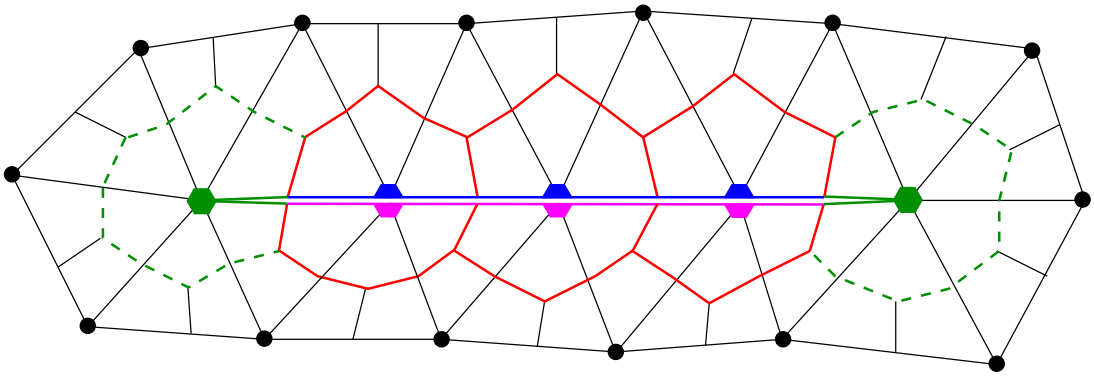


Figure 10.4: Dualgrid cells of a cut section through the disk.

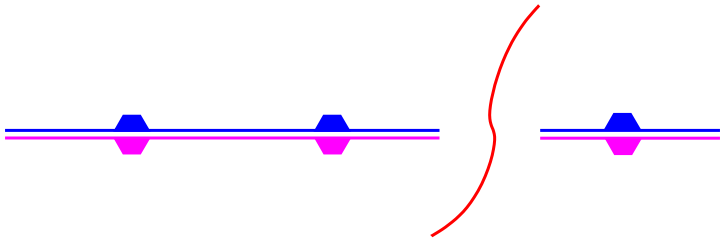


Figure 10.5: Partitioning of actuator disk periodic boundary.

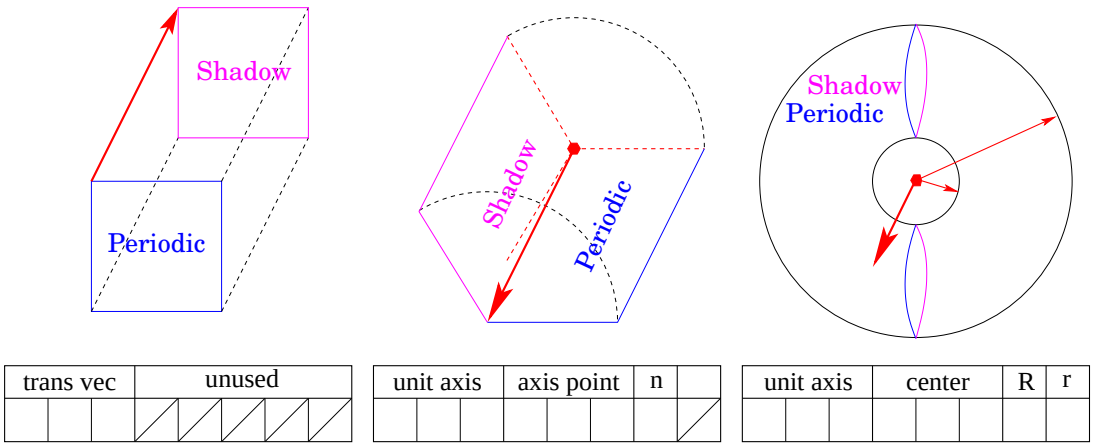


Figure 10.6: Geometry data of periodic types.

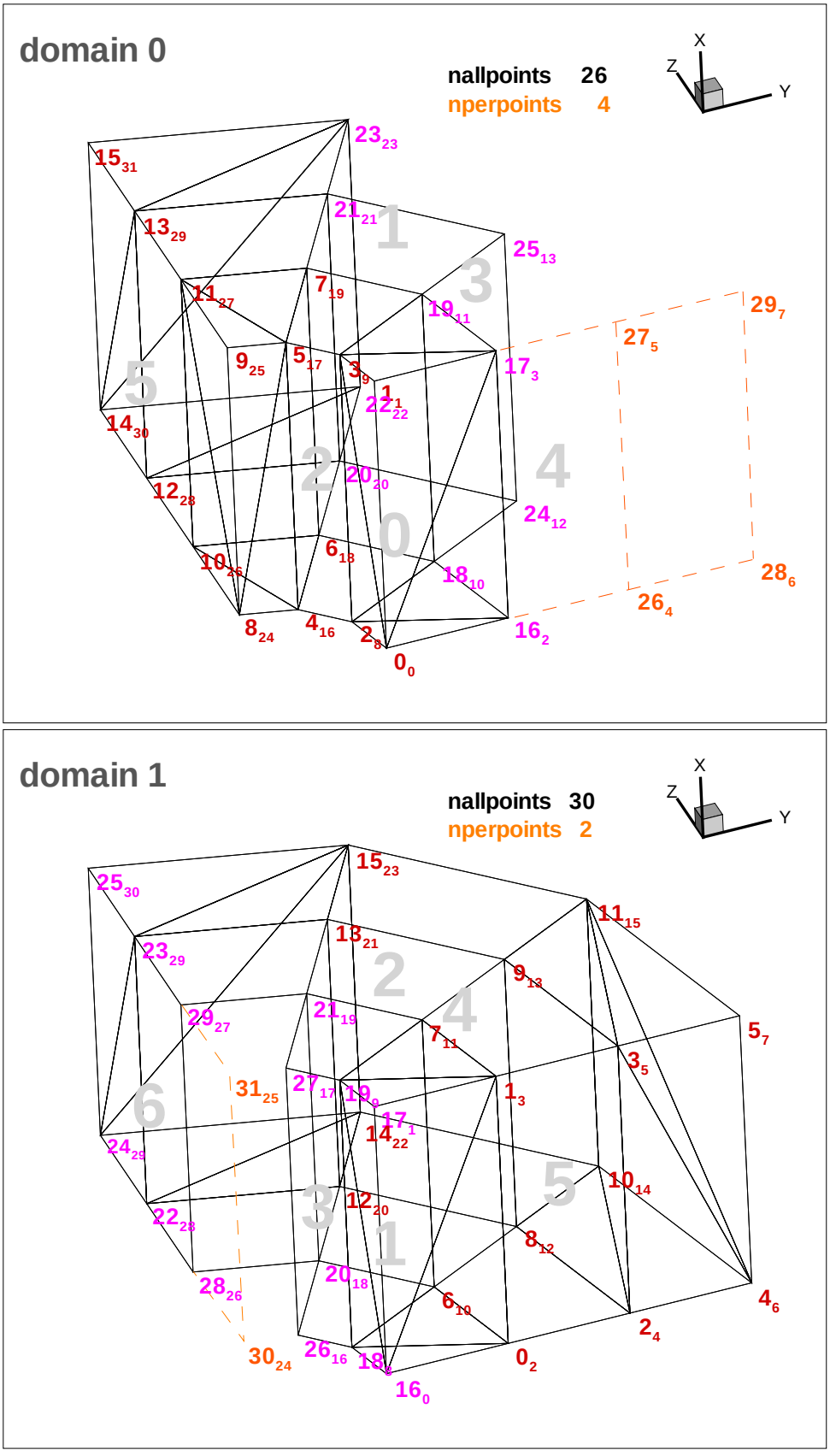


Figure 10.7: Partitioned grid with two periodic boundaries.

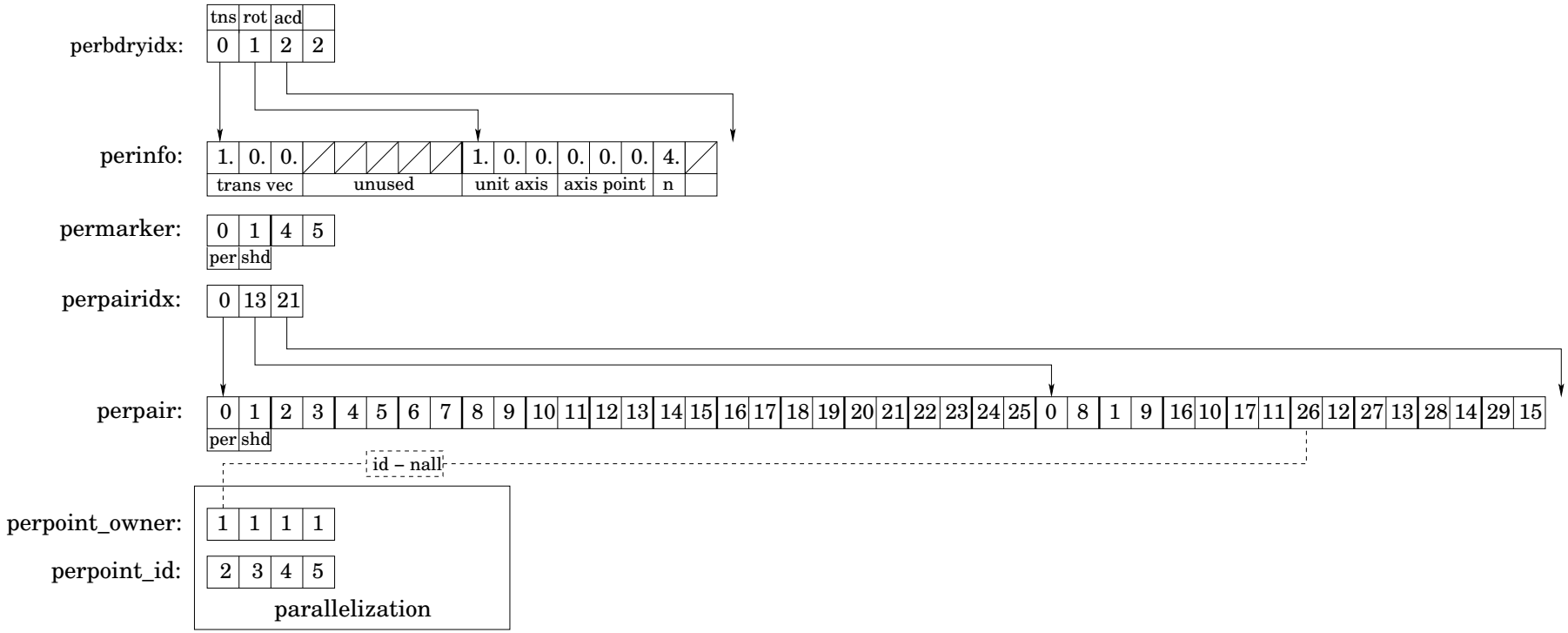


Figure 10.8: NetCDF format of periodic boundaries of domain 0.

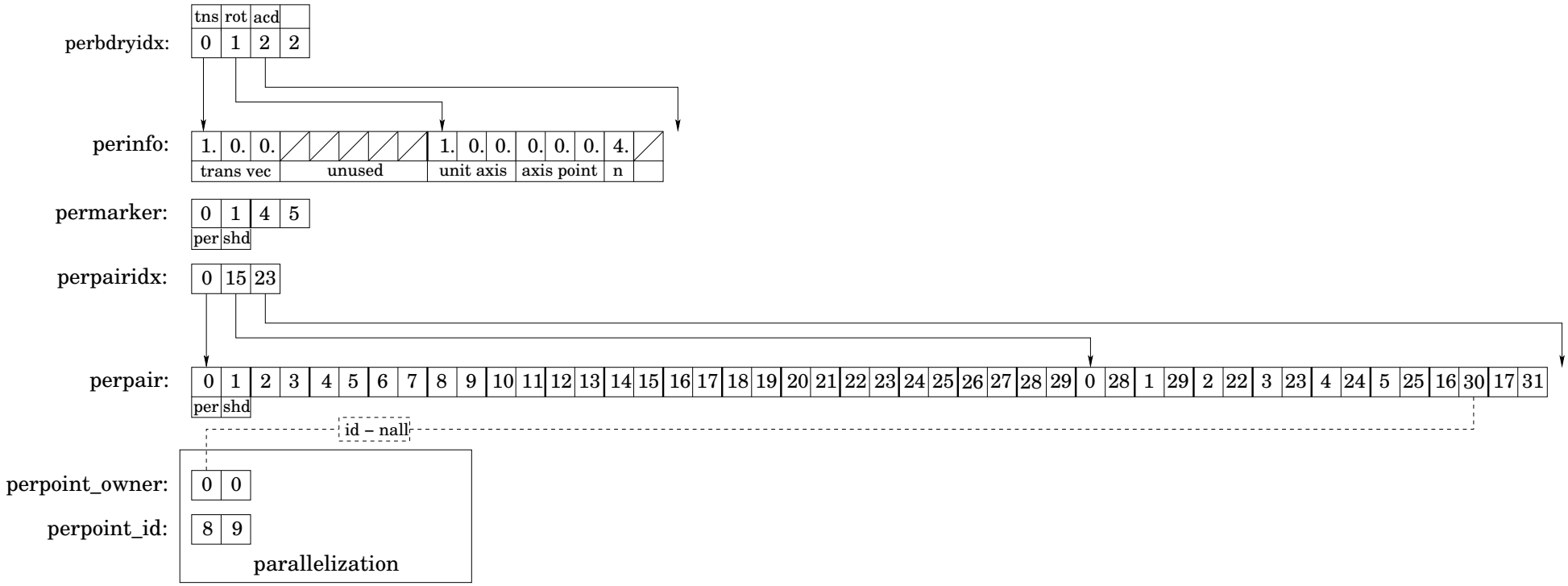


Figure 10.9: NetCDF format of periodic boundaries of domain 1.

11 Intermesh

11.1 Preface

This document provides detailed information about the implementation of the addon tool intermesh. For general information about features and usage the TAU user guide should be read.

11.2 Blockwise interpolation of point coordinates

The addon tool intermesh treats each chimera block of the grid on its own to provide the possibility of independent interpolation settings. Therefore it is possible to set up several chimera blocks for each flap/slat of a wing in the pregenerated grids and interpolate the new grid with independent settings of the flaps/slats while the main wing remains its shape. If more pregenerated grids are available for interpolation, only the most appropriate grids (i.e. smallest differences of the motion state parameter) are selected for the interpolation process. For different settings of the grid blocks the set of most appropriate grids may be different. Therefore the set of pregenerated grids is separately chosen for the interpolation process of each block .
However, the first given pregenerated grid is used as a template of the grid data for the preliminary setup of the new grid. To interpolate the points coordinates for each chimera block independently, it is necessary to loop over the blocks of the grid.

11.3 Parallelization

General aspects about the parallelization of intermesh In the process of subdividing a grid into several partitions for a parallel execution of the flow solver every partition needs additional points besides the distributed original grid points. These original points are the so-called ownpoints of the domain. The additional consist of the addpoints, extpoints and parpoints of the domain and are generally called remotepoints from now on. These points are also sorted according to their membership to the chimera blocks and have to be considered in the process of generating the new grid, too. Two examples of small partitioned grids and their corresponding sequence of global ids int the primgrid structure are shown in Figure 11.1.

The general sequence of entries in the primgrid array `global_id` and therefore the sequence of point coordinates is shown in Figure 11.2.

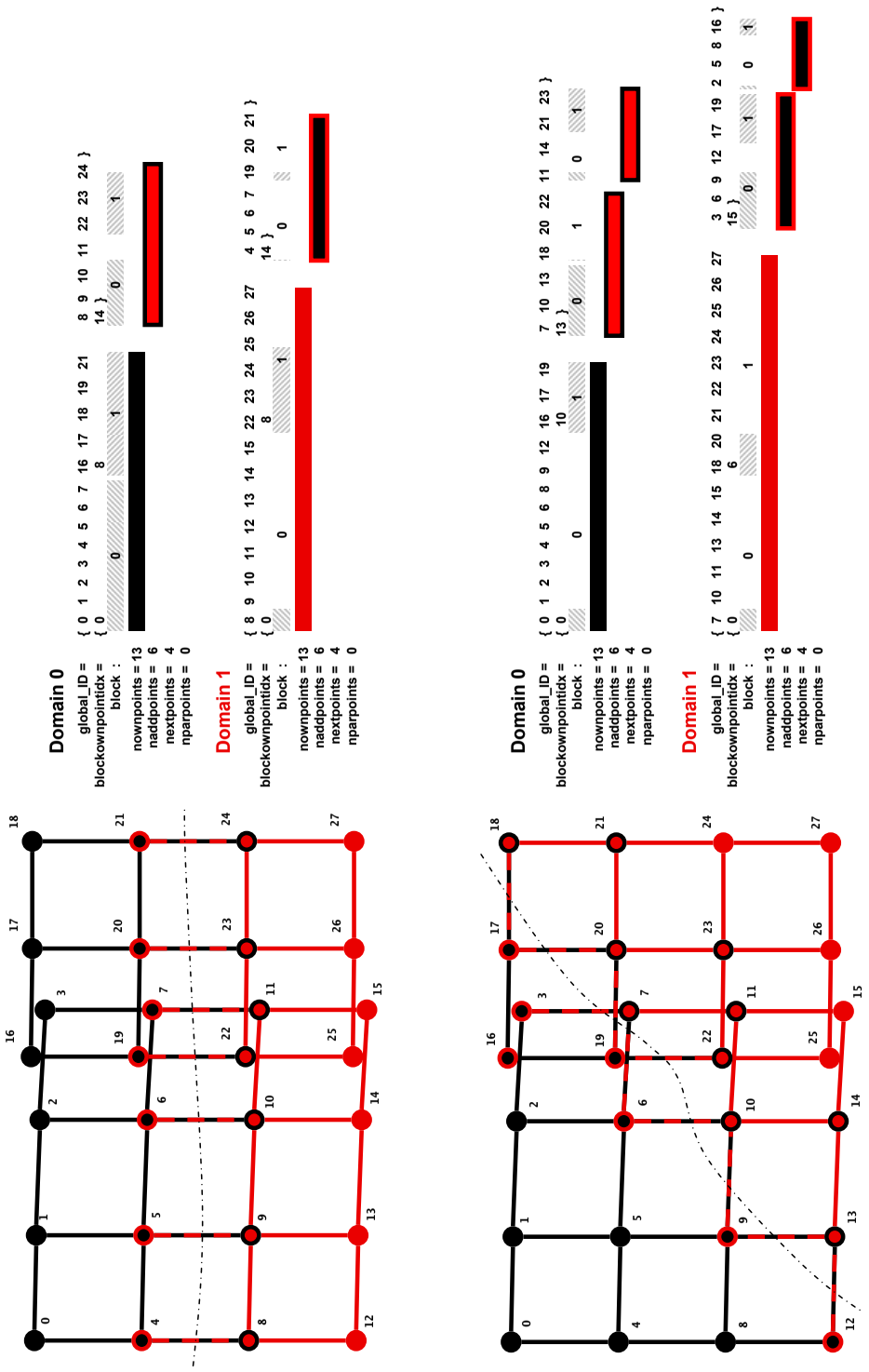


Figure 11.1: Primgrid pointdata arrays of partitioned grids

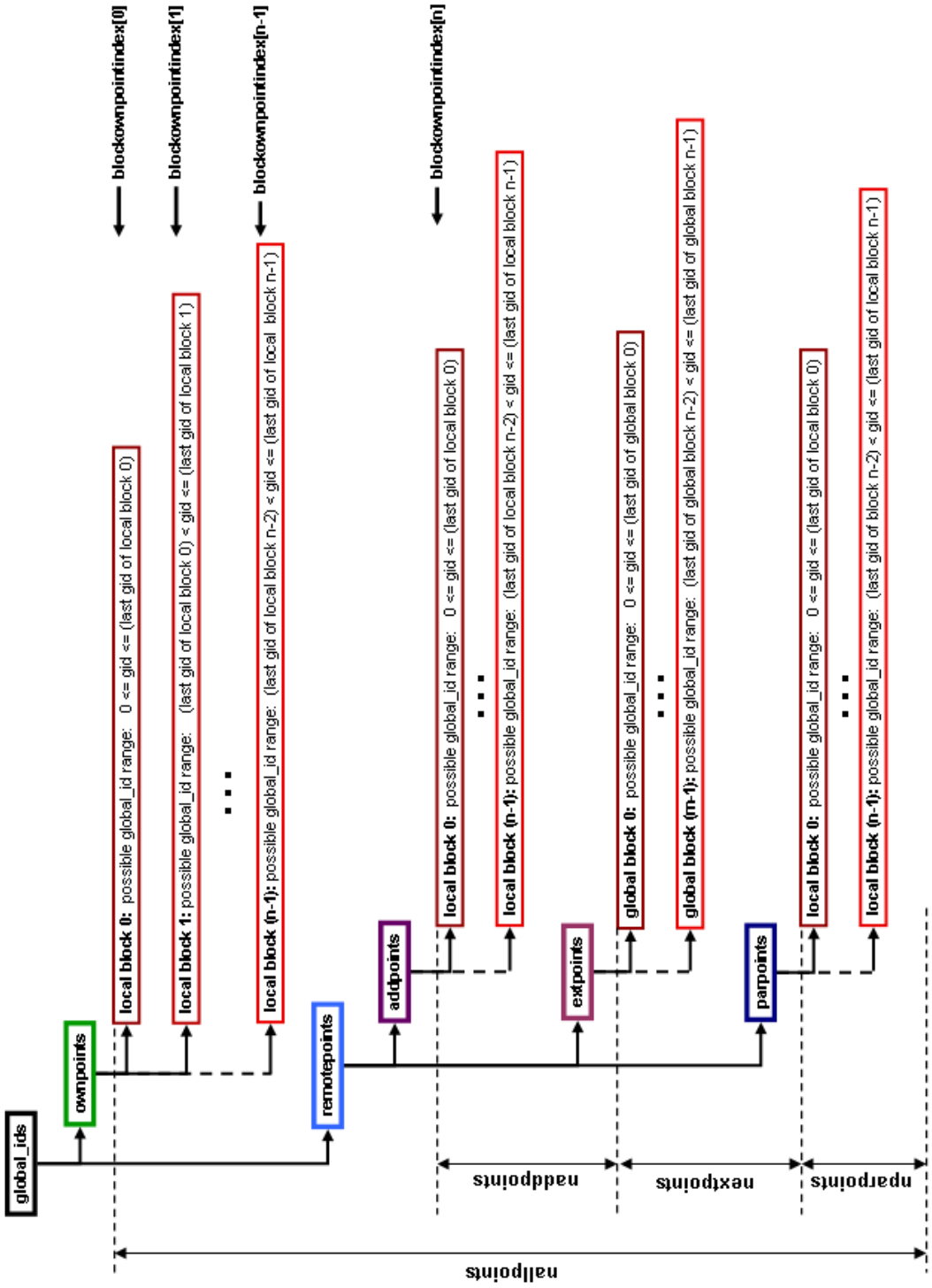


Figure 11.2: Sequence of global ids corresponding to block membership

11.3.1 Similar domain boundaries in all pregenerated grids

Interpolation of ownpoints

In the case of equally subdivided pregenerated grids the coordinate interpolation of the ownpoints is quite simple. The chimera block limits are given by the values of the array `blockownpointindex` and these limits are valid for all pregenerated grid domains assigned to a process. Therefore the interpolation process performs an outer loop over the present local blocks and an inner loop over the local points of the current block. Since there is no communication to other domains, each domain can be processed in parallel.

Interpolation of remotepoints

The interpolation of the remotepoints is almost as simple as the interpolation of the ownpoints for equal domain boundaries. Actually this means that the coordinates of the remotepoints are interpolated twice, since these coordinates have already been interpolated on the processes where they reside as ownpoints. Therefore it would be an alternative to update the coordinates of the remotepoints by communication rather than interpolating them again. However, communication is more expensive in terms of wall time, so the amount of communication is minimized. The only necessary communication left is to determine the block limits within the groups of remotepoints. This is done by searching the lowest and highest value of global ids in the current block throughout all domains. After this procedure, the process of interpolation can run in parallel without communication as well.

Interpolation of both ownpoints and remotepoints are performed in the outer loop to avoid unnecessary reading of grid files.

11.3.2 Different domain boundaries in pregenerated grids

Interpolation of ownpoints

In the case of different domain boundaries throughout the pregenerated grids, the process of interpolation cannot be accomplished in parallel, because corresponding grid points may belong to grid domains that are not assigned to the same process. Therefore it is necessary to loop over all points of a certain chimera block and gather the coordinates of the corresponding points from all pregenerated grids. In this case the grids are distributed to the number of precesses but the process of interpolation itself is performed sequentially.

Update of remotepoints

In this case the amount of communication needed to gather the necessary coordinates for the interpolation of one point is even larger than it is to update the coordinates from the process that owns that point. Therefore the coordinates of the remotepoints are updated instead of interpolated again.

Since the update process is independent from the pregenerated grids and the calculation of the block-dependent motion state parameter, it is performed after the outer loop over the chimera blocks.

12 Grid Quality Improvement

12.1 SmoothTaugrid

The program *smooth_taugrid* is developed to improve the quality of the unstructured part of a hybrid mesh. The main features are

- repairing of negative elements,
- to improve the quality of elements,
- to create anisotropic meshes.

The modifications are always lomathcally and the surface is untouched (except symmetry planes which are optionally allowed). The modified elements are prisms for 2D meshes and tetraeder and pyramids for 3D meshes.

12.1.1 Quality measures

For optimizing a mesh a quality measure for its elements has to be defined. The properties of the quality function should be that the measure is unity for an equilateral element and zero for a degenerated element. Values between zero and unity indicates valid non equilateral elements. Higher quality values should indicates a better quality of the element. Negative values appears if the volume (3D) or the surface (2D) of the element is negative in sense of the grid definition.

The used quality measure in the program *smooth_taugrid* based on the so mathcalled *mean ratio*

$$q_i = \begin{cases} 4\sqrt{3} \cdot \dfrac{A_i}{\sum_{j=1}^{3} l_{ij}^2} & \text{for 2D (triangles)} \\ 12 \cdot \operatorname{sign}(V_i) \cdot \dfrac{\sqrt[3]{(3V_i)^2}}{\sum_{j=1}^{6} l_{ij}^2} & \text{for 3D (tetrahedron)} \end{cases}, \tag{12.1}$$

where A_i is the area, V_i the volume and l_{ij} the edge lengths of the element i (see Figure 12.1*a*). Basically the measure is a ratio between the volume (3D) or area (2D) and the edge lengths.

Regarding [17] the mean measure is an effective measure in the sense of grid quality and computational effort. More quality measures can be found in [64]. At least one should note that the size of an element is not included in this measure.

The implementation of the mean measure allows also to use lomathcal anisotropic metrics. This can be helpful if more information about the flow e.g. a preliminary solution is available. In this case the orientation of the elements to the lomathcal flow is considered. Due to this new metric the edge lengths, area and volume is measured in the space of the new metric $\mathcal{M}$. The size functions in the new metric are than given by

$$l_{ij}^{\mathcal{M}} = \sqrt{(\vec{x}_i - \vec{x}_j)^T \cdot \mathcal{M} \cdot (\vec{x}_i - \vec{x}_j)}, \tag{12.2}$$

$$A_i^{\mathcal{M}} = \sqrt{det(\mathcal{M})} \cdot A_i, \tag{12.3}$$

$$V_i^{\mathcal{M}} = \sqrt{det(\mathcal{M})} \cdot V_i. \tag{12.4}$$

A reasonable metric can be extracted of the Hessian of the lomathcal Mach number Ma_{local}

$$\mathcal{H} = \frac{\partial}{\partial x_i \partial x_j} \mathrm{Ma}_{local} \tag{12.5}$$

or the pressure

$$\mathcal{H} = \frac{\partial}{\partial x_i \partial x_j} p. \tag{12.6}$$

A valid metric must be positive definite. Therefore the new lomathcal metric is given by

$$\mathcal{M} = \mathbf{R} \cdot \begin{pmatrix} |\lambda_1| & 0 & 0 \\ 0 & |\lambda_2| & 0 \\ 0 & 0 & |\lambda_3| \end{pmatrix} \cdot \mathbf{R} \tag{12.7}$$

with the eigenvalue decomposition

$$\mathcal{H} = \mathbf{R} \cdot \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \cdot \mathbf{R}. \tag{12.8}$$

It should be noted that the isotropic and anisotropic quality differs. The isotropic quality is the

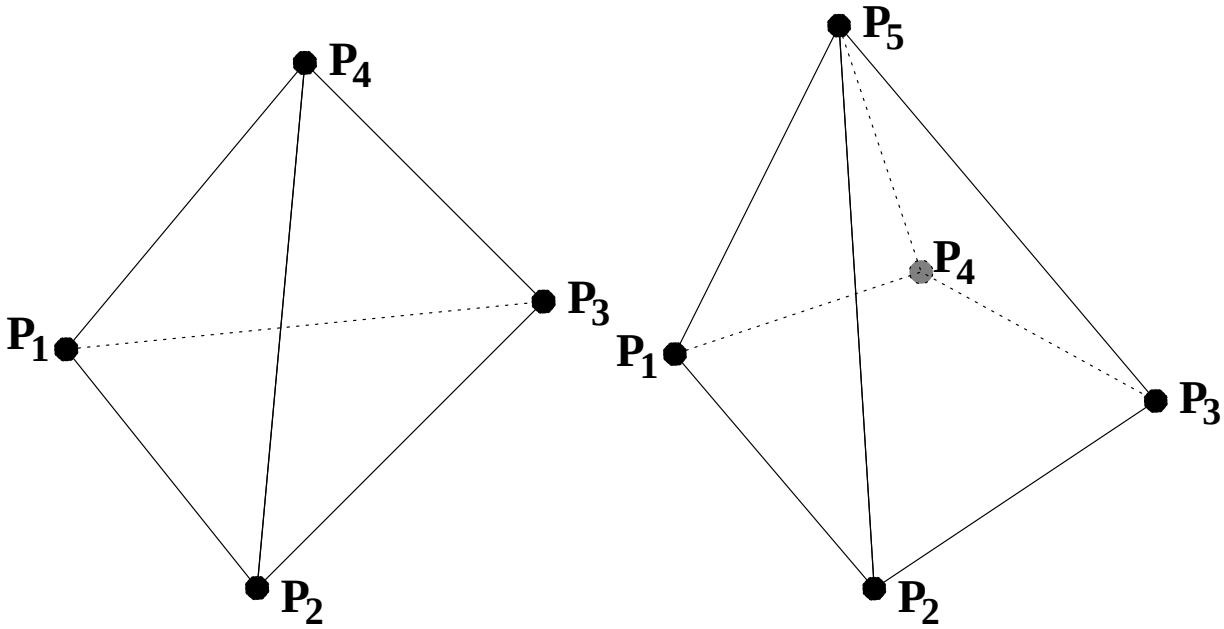


Figure 12.1: *(a)* Nomenclature of variables on a tetrahedral element *i*. *(b)* Sketch of the splitting method for pyramids.

best measure if no information about the flow is given and also if the mesh should be deformed. The anisotropic measure is useful if a solution is given.

As a second mesh quality indicator the minimal dihedral angle

$$\xi_i = \min_j(\phi_{ij}, 180^\circ - \phi_{ij}) \tag{12.9}$$

of tetrahedral elements (3D) and triangles (2D) is measured. Lower values are a potential risk for CFD computations and especially for deformation tools. The dihedral angle is always measured in the physimathcal space.

There exists also a quality measure for pyramids which is optional usable. The quality measure based of the definition for tetraeders. The pyramid will be split into four tetraeders (Figure12.1*b*) from which the quality q_i $(i \in 1,2,3,4)$ will be mathcalculated. The quality measure of the pyramid is the defined as

$$q_{i,pyramid} = \text{const.} \cdot \min_{j \in 1,2,3,4} q_j \tag{12.10}$$

The inserted midpoint of the base area is mathcalculated by a weightened method, similar to the method in the TAU code.

12.1.2 Goal functions

For pointwise optimization due to movement of a node a goal function has to be defined which combines the quality of surrounding elements S_i of a node P_i. In the program *smooth_taugrid* three different goal functions are defined

1. *least square* goal function
$$\bar{g}(S_i) = \frac{1}{N}\sqrt{\sum_{j \in S_i} q_j^2}$$
2. *inverse least square* goal function
$$\bar{g}(S_i) = \frac{N}{\sqrt{\sum_{j \in S_i} q_j^{-2}}}$$
3. *minimal quality* goal function
$$\bar{g}(S_i) = \min_{j \in S_i} q_j$$

The first two goal functions are continuously differentiable. The third one is only continuous but not continuously differentiable.

12.2 Methods for grid improvement

To improve the quality of a three-dimensional mesh four different methods are implemented which are described in the following subsections. For a two-dimensional grid only edge swapping and the movement of nodes are implemented.

12.2.1 Edge swapping

The edge swapping method changes the topology of the mesh due to deleting and inserting edges see [34, 25, 81, 27, 11, 12, 58]. In two-dimensional grids the edge swapping is implemented as shown in Figure 12.2*(a)*. Here the edge between two triangles is deleted and inserted at the former unconnected points.

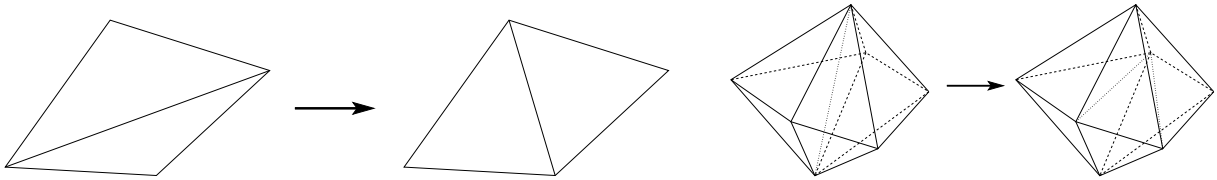


Figure 12.2: Examples for edge swapping for 2d *(a)* and 3d *(b)* meshes.

In three dimensions the algorithm is more complex and the possibilities of reconnecting the nodes are larger. An example is shown Figure 12.2*(b)*. Here the edge in the middle (dotted line in the left mesh) is deleted and two new edges are inserted (dotted lines in right mesh). This example shows that due to this manipulation the number of tetrahedrons is changing. Although there exist several possibilities to insert the new edges to get a valid mesh.

The implemented algorithm for three dimensional meshes based on the idea of the so called *equatorial edge swapping*. Outgoing from an edge surrounded by N_t tetrahedrons the edge swapping algorithm deletes this edge. After deleting the edge only a hull survives. In this hull new edges are inserted to make a valid grid of tetrahedrons within the hull. The number of possible combinations, edges and resulting tetrahedrons depends on the initial number of surrounding tetrahedrons N_t.

The general principle is shown in Figure 12.3*(a)*. Here an example for $N_t = 4$ is sketched but the principle holds also for $N_t \geq 3$. To apply the edge swapping the algorithm needs an edge which is surrounded only by tetrahedrons. In this case the outgoing edge connects the point 0 and 1. This edge will be removed in the first step of the algorithm. In the second step new edges will be placed on the pseudo equatorial plane, here spanned by the points 2–5. The number of inserted edges is given by $N_t - 3$.

In a projected view on the pseudo plane spanned by the points 2–5 one can schematically reduce the possibilities to the two-dimensional Figure 12.3*(b)*. Without swapping (left square) the polar edge between point 0 and 1 foraminate through the projected equatorial plane. All edges inserted by the swapping process are laying in this plane (middle squares in Figure 12.3*b*). In this example only one edge has to be inserted as a diagonal of the square. So they are only two different configurations possible. Both possible configurations are in principle the same and have the same pattern. They differ formally only by a rotation of the point indices. For convenience and for a general view this case reduces to a picture which is a square with a diagonal (right square in Figure 12.3*b*). This Figurerepresents then two combinations by rotation of the indices $(C_r = 2)$. Because the number of combinations increases dramatically with the number of surroundings N_t these simplified diagrams are necessary to get a better overview over all possible configurations for higher N_t. The number of combinations is given by

$$C_{total} = \frac{(2N_t - 4)!}{(N_t - 1)!(N_t - 2)!} \tag{12.11}$$

[58]. The possible combinations of the equatorial swapping are shown in the Chapter 12.3 (Figure 12.7, 12.9) for $3 \geq N_t \geq 8$. The possible variations due to rotation is given by the variable C_r for each pattern. Because of the increasing number of combinations only cases for $N_t \leq 8$ are implemented. In the case of $N_t = 8$ this leads to $C_{total} = 132$ combinations.

12.2.2 Face swapping

The face swapping is the reverse of the edge swapping for $N_T = 3$. The method is shown in Figure 12.4. Here the face between two tetrahedrons is deleted and three new faces and one new

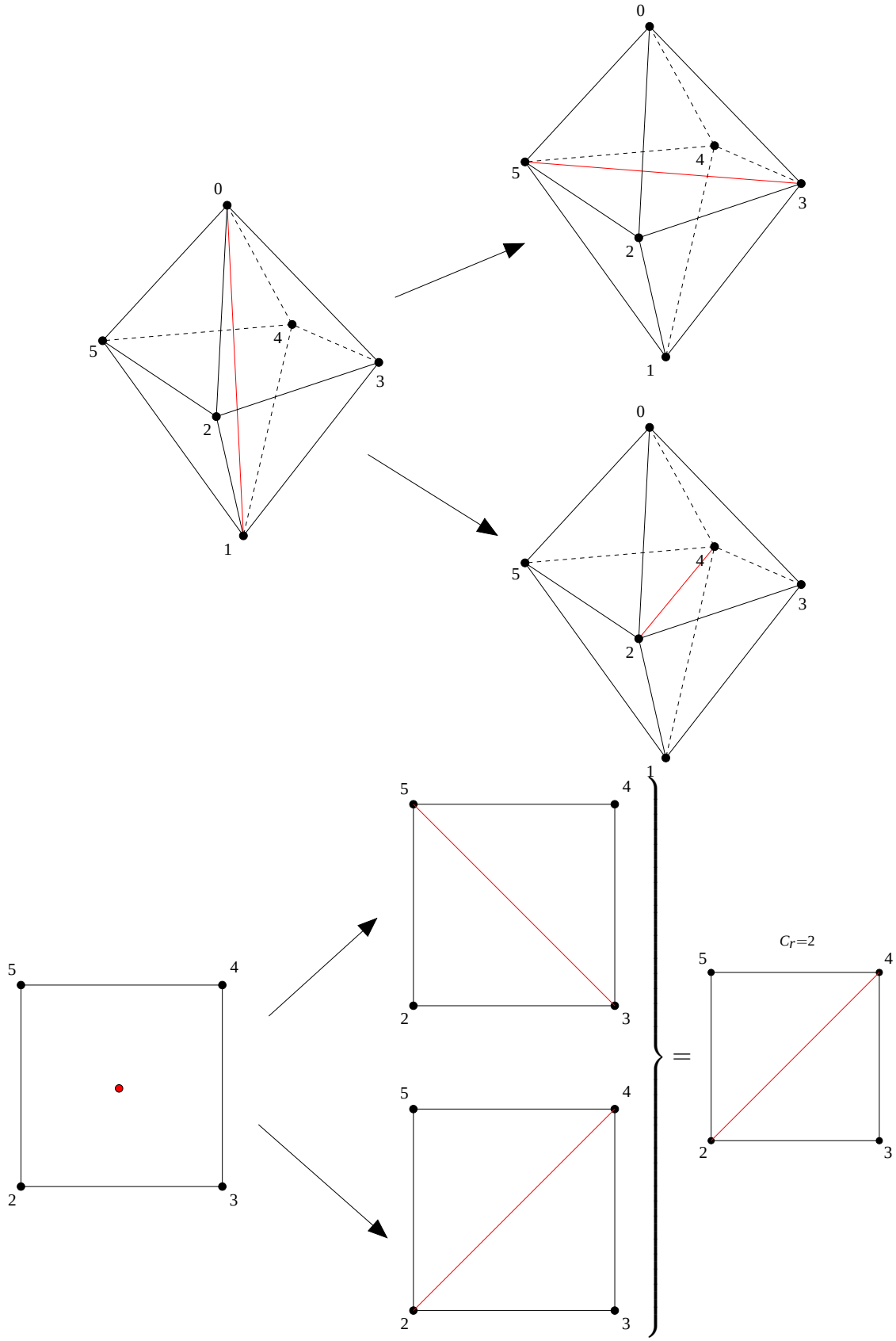


Figure 12.3: Principle of equatorial edge swapping for $N_T = 4$. *(a)* Three-dimensional view. *(b)* View of the projected pseudo plane spanned by the points $2-5$.

edge are inserted, respectively. Due to the manipulation the number of tetrahedrons increases by one. Although the implemented algorithm has the capability to do the face swapping on elements which are laying on the symmetry plane of a mesh. The face swapping is implemented

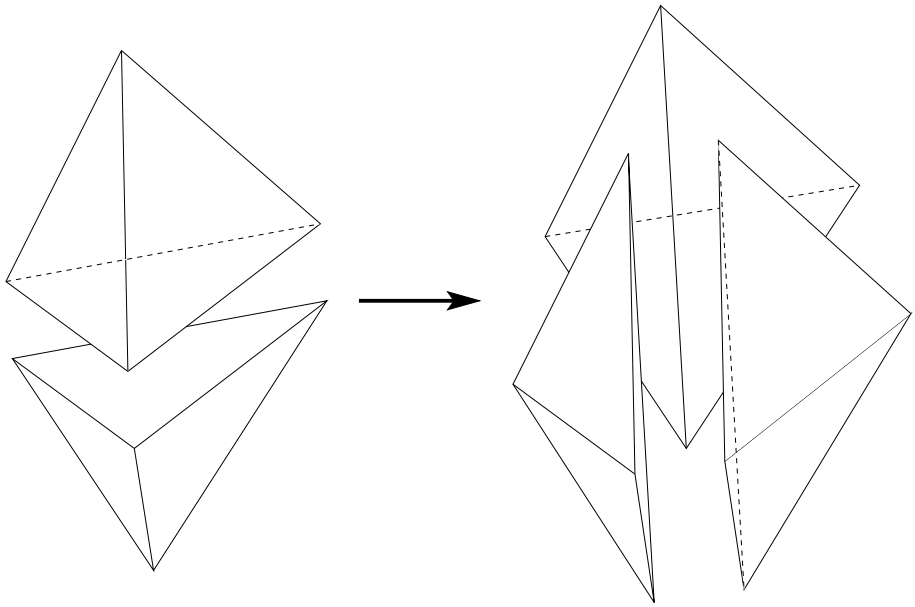


Figure 12.4: Schematic plot of a face swapping step.

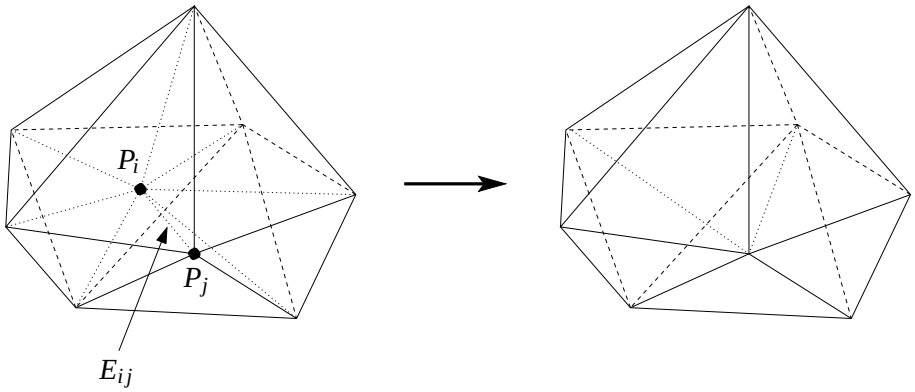


Figure 12.5: The edge collapsing degenerates the edge $E_{ij} = \overline{P_iP_j}$ and its surrounding elements.

only for 3D meshes.

12.2.3 Edge collapsing

The edge collapsing [40] changes the topology of a mesh by deleting an edge and a point, respectively. Figure 12.5 sketches the method. Starting from a point P_i which is surrounded of tetrahedrons the algorithm try to degenerate the edge $E_{ij} = \overline{P_iP_j}$ and its surrounding tetrahedrons and checks if the resulting mesh is improved. Due to this method the number of tetrahedrons decreases. The number of points decreases by one.

Additionally the algorithm is able to make an edge collapsing if the surrounding elements of point P_i are a mixture between tetrahedrons and pyramids. Note that the edge collapsing is only implemented for 3D meshes.

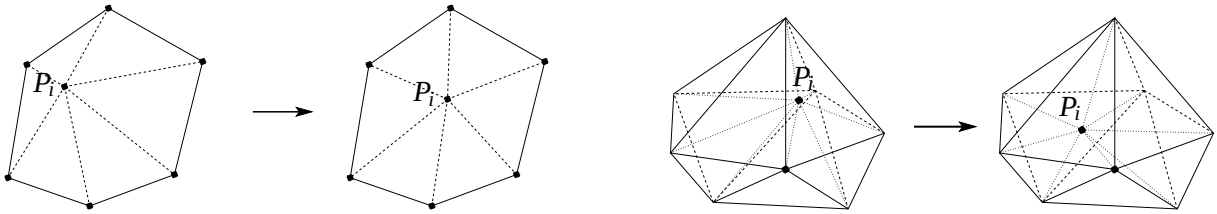


Figure 12.6: Sketch of the movement of a node in two *(a)* and three *(b)* dimensions.

12.2.4 Movement of nodes

The last strategy to improve a mesh is the movement of nodes. It pursue the strategy of the pointwise movement of nodes [25, 12][1]. A sketch of the method is shown in Figure 12.6. For finding the new position of a node a so called *combined method* is used [63, 12, 26]. This method combines the Laplacian smoothing and gradient based optimization methods. The Laplacian smoothing sets the new position of the node in the geometrical mean of its surrounding nodes. If this setting is not valid in the sense of quality improvement an gradient based optimizer tries to find a new position for the node. The choose of the gradient based optimization algorithm depends on the selected goal function. For a continuously differentiable goal function a conjugate gradient based method from the is used. For goal functions that are not continuously differentiable a so called *maxmin* optimizer was used. This algorithm is described in [25, 27] and [28]. Further information about this optimization algorithm can be found in [29] and [14].

The main reason for this combination is that the Laplacian smoothing is cheap in the sense of computational costs but very ineffective in three dimensions [25]. The computational cost of gradient based optimization methods are relatively expensive in comparison to the Laplacian smoothing. So this combination is a compromise between speed and improvement of the quality.

The combined algorithm is implemented for two- and three-dimensional grids. For three-dimensional grids it is also possible to move points on a symmetry plane and to move points which are connected to pinnacle of the pyramids.

12.2.5 Decisions for modifications

A modification of the grid is tried if it is geometrically possible and if the quality at least of one of the considered triangles (2d)/tetrahedrons and pyramids (3d) is less than the limit

$$q < q_{lim}. \tag{12.12}$$

Otherwise an optimization of these triangles (2d)/tetrahedrons and pyramids (3d) is skipped. If this criterion is fulfilled the modifications will be checked. To check if a change in the grid is valid several requirements has to be satisfied. The following demands has to be always fulfilled to accept a modification step:

- the minimal quality is larger than zero (to avoid inverted elements)
- the minimal quality is larger than the global minimal quality
- the goal function has to be improved (in the case of node movement)
- number of negative elements has to be decreased

[1] Another concept is the global optimization strategy which optimizes all points simultaneously [53, 54].

Additionally more restrictions can be selected optionally. The additional implemented restrictions are

- improve the most worse dihedral angle

$$\xi = \min_i \left(\min \left(\phi_i, 180^\circ - \phi_i \right) \right),$$

  where ϕ_i $(i = 1, \ldots, 4)$ are the dihedral angles of a tetrahedron

- improve the mean quality (and therefore the global quality of the mesh)
- the worst dihedral angle ξ should not be lower than a given limit ξ_{lim}

One should note that more restrictions defines higher demands on the quality but also reduces the possibility to repair worse elements. At least the restriction to improve always the dihedral is not useful for anisotropic grids because here stretched elements are wanted.

12.2.6 General algorithm

The general optimization is done by sequentially applying edge swapping, edge collapsing, face swapping and at least nodal optimization. This sequence can repeated several times due to an outer iteration loop. The main procedure is given by

1. evaluate the command line
2. read the grid
3. read the hessian from the solution file (in the case of anisotropic quality measures)
4. coarse check of the prisms (2d) and tetrahedrons (3d), respectively
5. initialisation tasks
   (e.g. computing the anisotropic metric and the initial quality, marking of movable points, additional output etc.)
6. outer optimization loop
   (a) loop of edge swapping
   (b) loop of edge collapsing
   (c) loop face swapping
   (d) loop of movement of nodes
7. store the optimized grid
8. reading the solution (if given)
9. interpolate the solution on the optimized grid
10. store the solution for the optimized grid

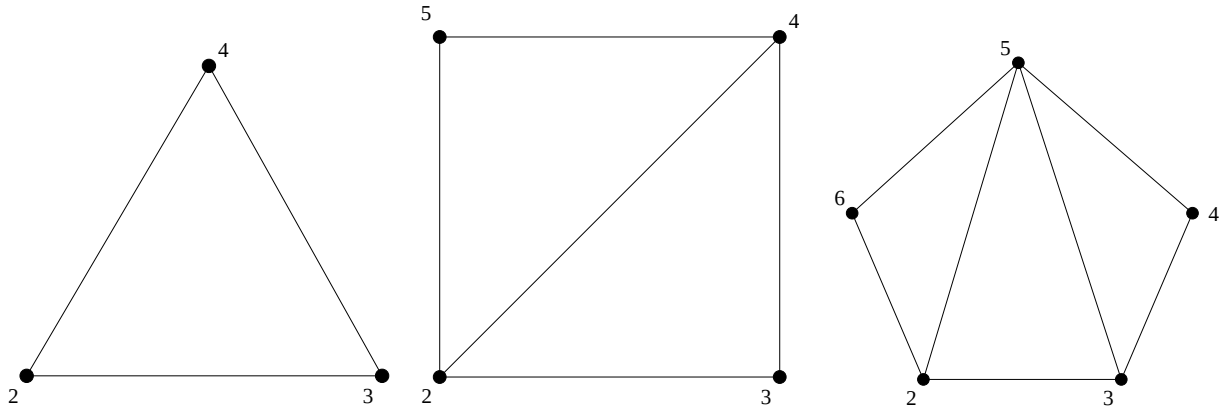
If the option `--force` is used to enable the repairing of negative elements the inner loop is modified. In this case the first three modifications (edge/face swapping, edge collapsing) are skipped for the first iteration so that the loops begins with a movement of nodes. This setting comes from first experiences of repairing meshes with negative elements.

Additionally some output of the statistic of the grid can be given optionally.

To improve the performance of *smooth_taugrid* the program uses intensively lists. These lists are containing information about edges, faces, surrounding elements of a point, changes in the grid, etc.

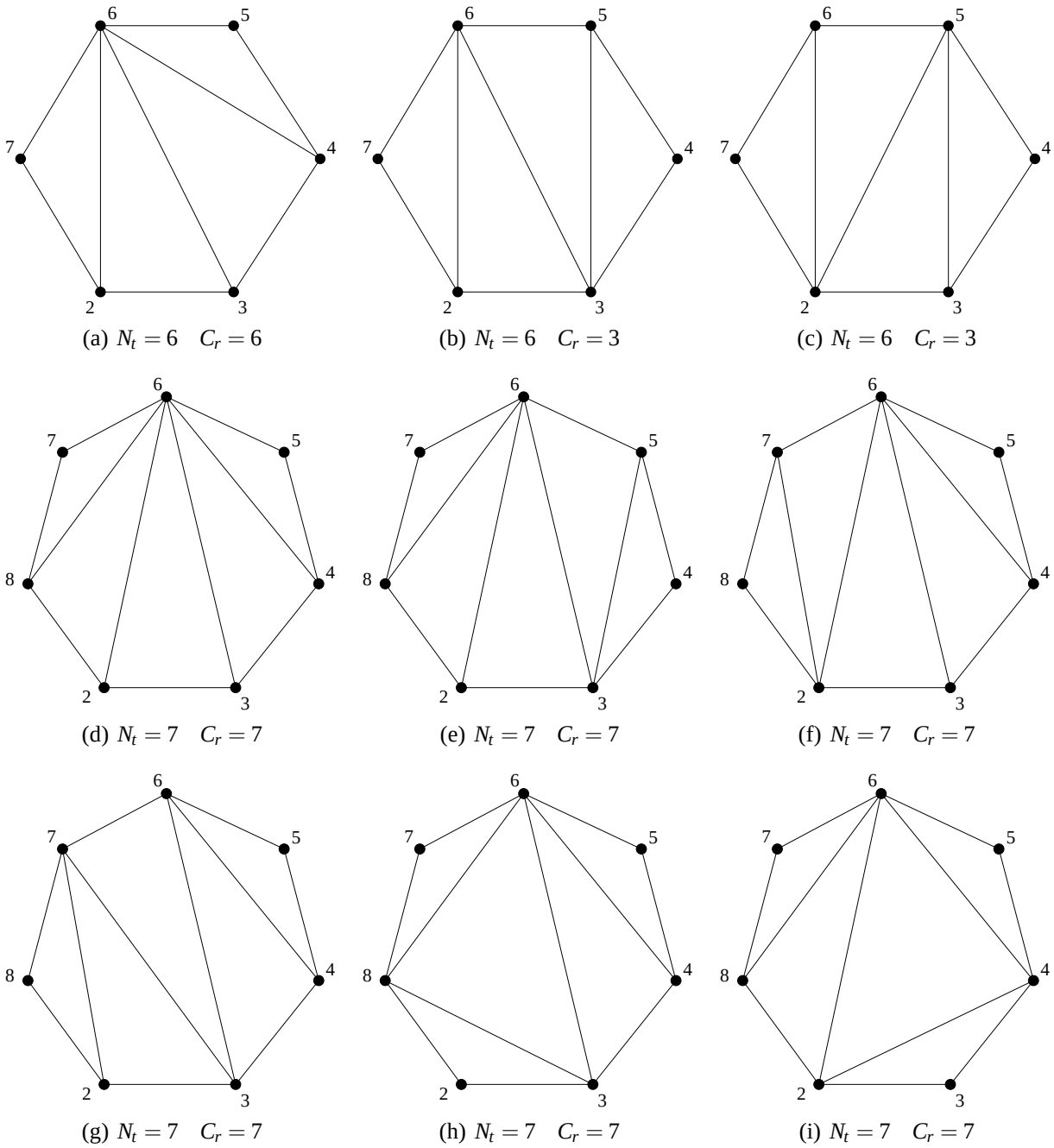
12.3 Configurations of equatorial edge swapping

Following pictures shows the possible configuration for the new edges in the equatorial plane. N_t denotes the number of surrounding tetrahedrons of the initial edge. The number of possible variations of a pattern by rotation is specified by the variable C_r.



(a) $N_t = 3 \quad C_r = 1$ (b) $N_t = 4 \quad C_r = 2$ (c) $N_t = 5 \quad C_r = 5$

Figure 12.7: Equatorial edge swapping for $N_t = 3$, $N_t = 4$ and $N_t = 5$.

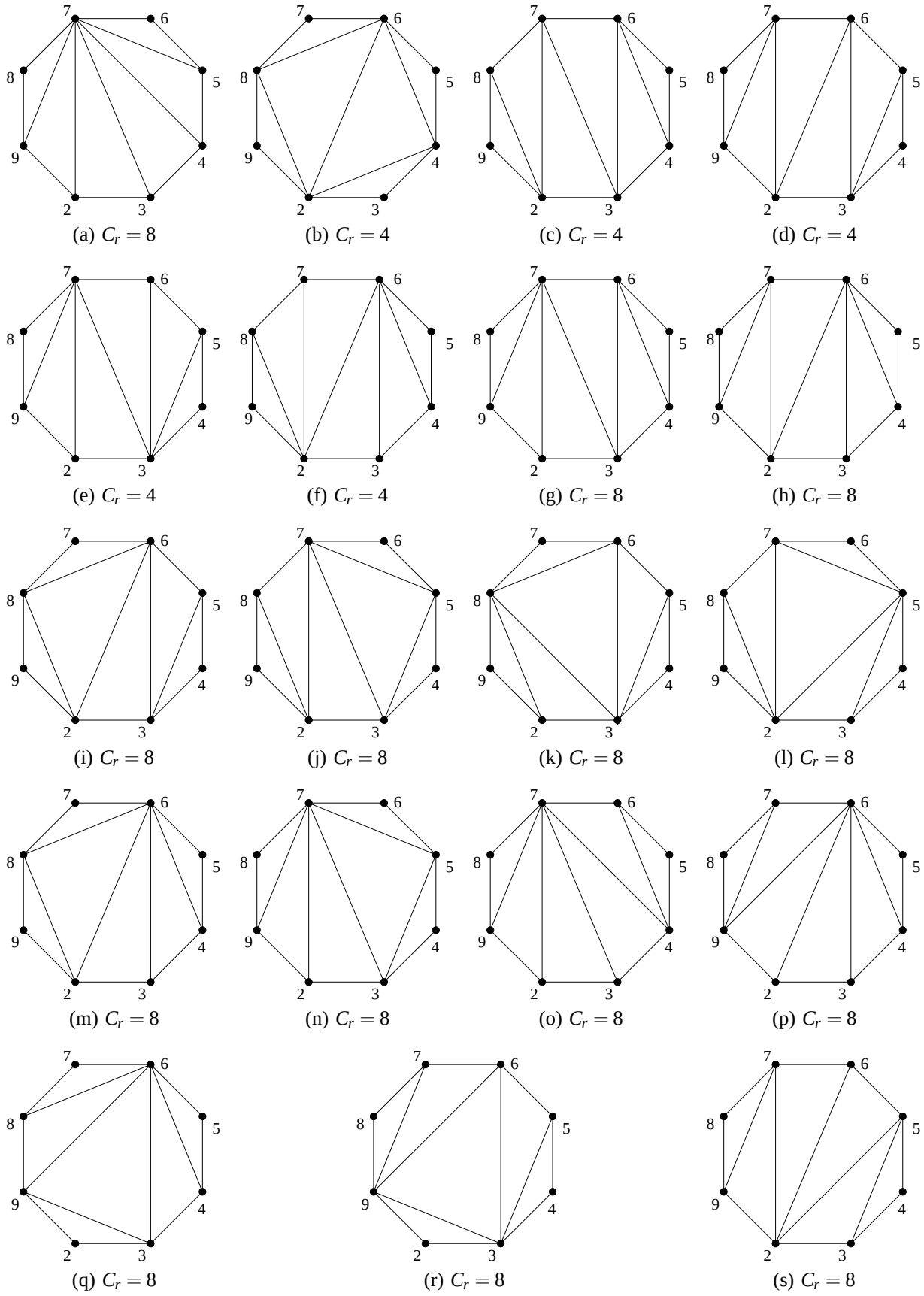


(a) $N_t = 6 \quad C_r = 6$ (b) $N_t = 6 \quad C_r = 3$ (c) $N_t = 6 \quad C_r = 3$

(d) $N_t = 7 \quad C_r = 7$ (e) $N_t = 7 \quad C_r = 7$ (f) $N_t = 7 \quad C_r = 7$

(g) $N_t = 7 \quad C_r = 7$ (h) $N_t = 7 \quad C_r = 7$ (i) $N_t = 7 \quad C_r = 7$

Figure 12.8: Equatorial edge swapping for $N_t = 6$ and $N_t = 7$.



(a) $C_r = 8$ (b) $C_r = 4$ (c) $C_r = 4$ (d) $C_r = 4$

(e) $C_r = 4$ (f) $C_r = 4$ (g) $C_r = 8$ (h) $C_r = 8$

(i) $C_r = 8$ (j) $C_r = 8$ (k) $C_r = 8$ (l) $C_r = 8$

(m) $C_r = 8$ (n) $C_r = 8$ (o) $C_r = 8$ (p) $C_r = 8$

(q) $C_r = 8$ (r) $C_r = 8$ (s) $C_r = 8$

Figure 12.9: Equatorial edge swapping for $N_t = 8$.

Bibliography

[1] *Rew, R. and Davis, G. and Emmerson, S.*, NetCDF User's Guide, Unidata Program Center, 1993.

[2] *Anderson W. Kyle, Bonhaus Daryl L.* , An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids, Computers Fluids Vol. 23, No. 1, pp 1-21, 1994.

[3] I. Babuška, A. K. Aziz, Survey lectures on the mathematical foundations of the finite element method. In the Mathematical Foundation of the Finite Element Method with Applications to Partial Differential Equations, A. K. Aziz, ed., Academic Press, New York - London, pp. 3-363, 1972.

[4] *Barth, T.J. and Jesperson, D.C.*, The design and application of upwind schemes on Unstructured Meshes, AIAA-89-0366, 1989.

[5] *Beam, Richard M. and Warming, R.F.*, An Implicit Finite-Difference Algorithm for Hyperbolic Systems in Conservation-Law Form, Journal of Computational Physics 22, pp.87-110, 1976.

[6] *Brandt, A.*, Multi-level adaptive techniques (MLAT). I. The multi-grid method, Research Report RC 6026, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1976.

[7] *Benek, J.A., Buning, P.G, Steger, J.L.*, A 3-D Chimera Grid Embedding Technique, AIAA 85-1523, 1985.

[8] *J. Bonet, J. Peraire.*, An alternating digital tree (ADT) algorithm for Geometric Searching and Intersection Problems, Int. J. Num. Meth. Eng, Vol. 31, pp. 1-17, 1991.

[9] *Brandt, A.*, Multi-level adaptive solutions to boundary value problems, Math. Comp., Vol. 31, pp. 333-390, 1977.

[10] *Brandt, A.*, Multigrid techniques: 1984 guide with applications to fluid dynamics, VKI lecture series 1984-04, 1984.

[11] *Buscaglia, G. C. and Dari, E. A.*, Anisotropic mesh optimization and its application in adaptivity. *Int. J. Numer. Meth. Engrg* **40**, pp. 4119-4136, 1997

[12] *Canann, S. A., Tristano, J. R. and Staten, M. L.*, An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In *7th International Meshing Roundtable, Dearborn, Michigan*, pp. 479-494. Sandia National Labs, 1998

[13] *Chen, H.C. and Yu, N.J. and Rubbert, P.E. and Jameson, A.*, Flow Simulations for General Nacelle Configurations Using Euler Equations, AIAA-83-0359, 1983.

[14] *Charalambous, C. and Conn, A.*, An efficient method to solve the minimax problem directly. *SIAM J. Numer. Anal.* **15**, pp. 162-187, 1978

[15] *Choi, Y.H. and Merkle, C.L.*, The Application of Preconditioning to Viscous Flows, Journal of Computational Physics 105, pp. 207-223, 1993.

[16] *Chorin, Alexandre Joel*, A Numerical Method for Solving Incompressible Viscous Flow Problems, Journal of Computational Physics 2, pp. 12-26, 1967.

[17] *Dompierre, Vallet, Labbé, Guibault*, An analysis of simplex shape measures for anisotropic meshes, Comput. Methods in Appl. Mech. Engrg **64**, pp. 4895-4914.

[18] *Dougherty, F.C.*, Development of a Chimera Grid Scheme with Applications to Unsteady Problems, Ph.D Dissertation, Stanford University, April 1985.

[19] *Dougherty, F.C., Benek, J.A., and Steger, J.L.*, On Applications od Chimera Grid Schemes to Store Separation, NASA TM88193, October 1985.

[20] *Dwight, Richard P.*, Efficiency improvements of RANS-based analysis and optimization using implicit and adjoint methods on unstructured grids, University of Manchester, 2006.

[21] *Edwards, J.R., Chandra, S.*, Comparison of Eddy Viscosity-Transport Turbulence Models for Three-Dimensional, Shock-Separated Flowfields, AIAA Journal, Vol. 34, No. 4, April 1996.

[22] *Eisfeld, B.* Implementation and validation of the Hellsten k-ω EARSM, DLR IB 124-2003/33, 2003.

[23] *Turkel, E.*, Improving the Accuracy of Central Difference Schemes, ICASE-report No. 88-53, 1988.

[24] *Franke, M.* Untersuchung zum Potential höherwertiger Turbulenzmodelle für den aerodynamische Entwurf, Dissertation Technische Universität Berlin, 2003.

[25] *Freitag, L. A. and Ollivier-Gooch, C.*, A comparison of tetrahedral mesh improvement techniques. In *Proceedings of the Fifth International Meshing Roundtable*, pp. 87-100, Albuquerque, NM. Sandia National Laboratories.

[26] *Freitag, L. A.*, On combinig Laplacian and optimization-based mesh smoothing techniques. *ASME* **220**, pp. 37-43, 1997

[27] *Freitag, L. A. and Ollivier-Gooch, C.*, Tetrahedral mesh improvement using swapping and smoothing. *Int. J. Numer. Meth. Engrg* **40**, pp. 3979-4002.1997

[28] *Freitag, L., Jones, M. and Plassmann, P.*, A parallel algorithm for mesh smoothing. *SIAM J. Sci. Comput.* **20**, pp. 2023–2040, 1999

[29] *Gill, P. E., Murray, W. and Wright, M. H.*, Practical optimization. Academic Press, London, UK, 1981

[30] *Hellsten, A.*, New Two-Equation Turbulence Model for Aerodynamics Applications, PhD thesis, Helsinki University of Technology, Laboratory of Aerodynamics, Finnland, 2004.

[31] *Haselbacher, A.; Blazek, J.*, On the Accurate and Efficient Discretisation of the Navier-Stokes Equations on Mixed Grids, AIAA J. 99-33552, 1998.

[32] *Jameson, A.; Schmidt, W.; Turkel E.*, Numerical Solutions of the EulerEquations by the Finite Volume Methods Using Runge Kutta Time Stepping Schemes, AIAA81-1259, 1981.

[33] *Jameson, A.*, Transonic Flow Calculations, MAE Report 1651, Princeton University, Princeton, New Jersey, 1983.

[34] *Joe, B.*, Three-dimensional triangulations from local transformations. SIAM J. Sci. Stat. Comput. **10**, pp. 718-741.

[35] *Kroll, N.; Jain, R. K.*, Solution of Two-Dimensional Euler Equations - Experience with a Finite Volume Code, DLR-FB 87-41, 1987.

[36] *Kroll, N.; Radespiel, R.*, An improved flux vector split discretisation scheme for viscous flows, DLR-FB 93-53, 1993.

[37] *Hänel, D.; Schwane, R.*, An implicit flux-vector splitting scheme for the computation of viscous hypersonic flow, AIAA Paper 89-0247, 1989.

[38] *Kok, J. C.* , Resolving the Dependence on Freestream Values for the k/ω Turbulence Model, AIAA Journal, Vol. 38, No. 7, p. 1292-1295, 2000.

[39] *Kok, J. C.* , Extra-large eddy simulation of massively separated flows, , AIAA Journal, Vol. 38, No. 7, AIAA-paper 2004-264, 2004.

[40] *Li, X., Shephard, M. S. and Beall, M. W.*, 3D anisotropic mesh adaptation by mesh modification, *Comput. Methods in Appl. Mech. Engrg* **194**, pp. 4915-4950, 2005

[41] *Liou, M. S.; Steffen, C. J.*, A new flux splitting scheme, J.Comp.Phy Vol. 107, pp. 23-39, 1993.

[42] *Madrane, A., Heinrich R. and Gerhold T.*, Implementation of the Chimera method in the unstructured hybrid DLR finite volume Tau-Code, 6th Overset Composite Grid and Solution Technology Symposium, Ft. Walton Beach, Florida, USA, October 8-10, 2002. http://www.arl.hpc.mil/Overset2002/.

[43] *Madrane, A.*, Parallel Implementation of a Dynamic Unstructured Chimera Method in the DLR Finite Volume TAU-Code, Accepted for the Third International Conference on Computational Fluid Dynamics, to take place in Toronto, Canada, http://oddjob.utias.utoronto.ca/iccfd3,July 12-16,2004.

[44] *Madrane, A.*, Parallel Implementation of a Dynamic Overset Unstructured Approach, Accepted for ECCOMAS2004, to take place in Jyvaskyla, Finland, http://www.mit.jyu.fi/eccomas2004,July 24-28,2004.

[45] *Martinelli, L.; Mavriplis, D.J.*, Validation of a Multigrid Method for the Reynolds Averaged Equations, AIAA-88-0414, 1988.

[46] *Mavriplis, D.J.*, Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes, AIAA-88-3706-CP, 1988.

[47] *Mavriplis, D.J.; Jameson, A.; Martinelli, L.*, Multigrid solution of the Navier-Stokes Equations on triangular Meshes, ICASE-report No. 89-35, 1989.

[48] *Mavriplis, D.J.*, Adaptive Meshing Techniques for Viscous Flow Calculation on Mixed-Element Unstructured Meshes, AIAA-97-0857, 1997.

[49] *Meakin, R., and Suhs, N.*, Unsteady Aerodynamic Simulation of Multiple Bodies in Relative Motion, AIAA 89-1996. Presented at the 9th AIAA Computation Fluid Dynamics Conference, Buffalo, NY, June 1989.

[50] *Meakin, R.*, Unsteady Simulation of the Viscous Flow about a V-22 Rotor and Wing in Hover, AIAA 95-3463, Presented at the AIAA Atmospheric Flight Mechanics Conference, Baltimore, MD, Aug. 1995.

[51] *Menter, F.*, Zonal Two Equation k-ω Turbulence Models for Aerodynamic Flows, AIAA 93-2906, 1993

[52] *Menter, F.*, Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications, AIAA Journal, Vol. 32, No. 8, August 1994

[53] *Munson, T. S.*, Mesh shape-quality optimization using the inverse mean-ratio metric. Technical Report ANL/MCS-P1136-0304, Argonne National Laboratory, 2004

[54] *Munson, T.*, Optimizing the quality of mesh elements. Technical Report ANL/MCS-P1260-0605, Argonne National Laboratory, 2005

[55] *Nakahashi, K., Togashi, F., Sharov, D.*, An Intergrid-Boundary Definition Method for Overset Unstructured Grid Approach, AIAA J. vol. 38, No. 11, pp. 2077-2084, 2000.

[56] *Radespiel, R.; Rossow, C.-C.*, A Cell Vertex Finite Volume Scheme for the Two-dimensional Navier-Stokes Equations, DLR-IB 129-87/40, 1987

[57] *Radespiel, R. and Turkel, E. and Kroll, N.*, Assessment of Preconditioning methods, DLR Forschungsbericht, 1995.

[58] *Pain, C. C., Umpleby, A. P., de Oliveira, C. R. E. and Goddard, A. J. H.*, Tetraherdral mesh optimization and adaptivity for steady-state and transient finite element calculations. *Comput. Methods in Appl. Mech. Engrg* **190**, pp. 3771-3796, 2001

[59] *Roe, P.L.*, Approximate Riemann Solvers Parametric Vectors and Difference Schemes, Journal of Computational Physics, Vol. 43, pp. 357-372, 1981

[60] *Rudnik, R.*, Erweiterung eines dreidimensionalen Euler-Verfahrens zur Berechnung des Strömungsfeldes um Nebenstromtriebwerke mit Fan- und Kernstrahl, DLR-FB 91-13, 1991

[61] *Rung, T.; Lübcke, H.; Franke, M.; Xue, L.; Thiele, F.; Fu, S.* Assessment of Explicit Algebraic Stress Models in Transonic Flows, Engineering Turbulence Modelling and Experiments 4, Proc. 4th Int. Symposium on engineering Turbulence Modelling and Measurements, Corsica, France, pp.659-668, Elsevier, Amsterdam, 1999.

[62] *Rung, T.; Bunge, U.; Schatz, M.; Thiele, F.* Restatement of the Spalart-Allmaras Eddy-Viscosity Model in a Strain-Adaptive Formulation, AIAA Journal 41(7), 1396-1399, 2003.

[63] *Shephard, M. S. and Georges, M. K.*, Automatic three-dimensional mesh generation by the finite octree technique. *Int. J. Numer. Meth. Engrg* **32**, pp. 709-749, 1991

[64] *Shewchuk, J. R.* What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures. University of California at Berkeley, unpublished preprint.

[65] *Sonar, T.; Süli E.*, A dual graph norm refinement indicator for the DLR- τ - Code, DLR-FB 94-24, 1994.

[66] *Spalart, P.R.; Allmaras, S.R.*, A One-Equation Turbulence Model for Aerodynamic Flows, AIAA-92-0439, 1992.

[67] *Spalart, P.R.; Jou, W.H.; Strelets, M.; Allmaras, S.R.*, Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach, First AFOSR International Conference on DNS/LES, Ruston, LA, 4-8, August, 1997, in: Advances in DNS/LES, edited by C. Liu and Z. Liu (Greyden, Columbus, OH, 1997).

[68] *Steger, J.L., Dougherty, F.C., and Benek, J.A.*, A Chimera Grid Scheme, ASME Mini-Symposium on Advances in Grid Generation, 1982.

[69] *Travin, A., Shur, M., Strelets, M., Spalart, P.R.*, Physical and numerical upgrades in the detached-eddy simulations of complex turbulent flows, In: Advances in LES of Complex Flows, p.239-254 (R. Friedrich and W. Rodi, eds.), Kluwer Academic Publishers, 2002.

[70] *Turkel, E.*, Preconditioned Methods for Solving the Incompressible and Low Speed Compressible Equations, Journal of Computational Physics 72, pp. 277-298, 1987.

[71] *Turkel, E and Fiterman, A. and van Leer, B.*, Preconditioning and the Limit to the Incompressible Flow Equations, NASA Contractor Report 191500, ICASE Report 93-42, 1993.

[72] *Venkatakrishnan, V.*, On the Accuracy of Limiters and Convergence to Steady State Solutions, AIAA 93-0880, 1993.

[73] *Venkateswaran, S. and Merkle, C.L.*, Analysis of Preconditioning Methods for the Euler and Navier-Stokes Equations, University of Tennessee Space Institute, 1999.

[74] *Viozat, Cécile*, Implicit Upwind Scheme for Low Mach Number Compressible Flows, Institut National de Recherche en Informatique et en Automatique, 1997.

[75] *Wada, Y.; Liou, M.S.* A flux splitting scheme with high-resolution and robustness for discontinuities, AIAA Paper 94-0083, 1994.

[76] *Wallin, S.; Johansson, A.V.* An explicit algebraic Reynolds stress model for incompressible and compressible turbulent flows, J. Fluid Mech. (403), pp. 89-132, 2000.

[77] *Wallin, S.* An efficient explicit algebraic Reynolds stress k/ω model (EARSM) for aeronautical applications, Technical report, FFA TN 1999-71 1999.

[78] *Weiss, J.M. and Smith, W.A.*, Preconditioning applied to Variable and Constant Density Time-Accurate Flows on Unstructured Meshes, 25th AIAA Fluid Dynamics Conference, 94-2209, 1994.

[79] *Whitfield, D.L.*, Three-dimensional unsteady Euler Equation solutions using flux vector splitting, 1983

[80] *Wilcox, D.C.*, Turbulence Modeling for CFD, 2bd ed. 1998, DCW Industries, La Cañada, California

[81] *Zavattieri, P. D., Dari, E. A. and Buscaglia, G. C.*, 1996 Optimization strategies in unstructured mesh generation. *Int. J. Numer. Meth. Engrg* **39**, pp. 2055-2071. 1996

List of Figures

1.1 Tetrahedron$_i = \{P_1, P_2, P_3, P_4\}$ . . . 2
1.2 Prism$_i = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ . . . 2
1.3 Pyramid$_i = \{P_1, P_2, P_3, P_4, P_5\}$ . . . 3
1.4 Hexahedron$_i = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$ . . . 3
1.5 Surface triangle$_i = \{P_1, P_2, P_3\}$ . . . 3
1.6 Surface quadrilateral$_i = \{P_1, P_2, P_3, P_4\}$ . . . 3
1.7 Boundary markers for a grid around an aircraft . . . 5
1.8 Control volume borders on a triangle . . . 8
1.9 Control volume borders on a quadrilateral . . . 9
1.10 Control volume borders in a tetrahedron . . . 9
1.11 Control volume borders in a prism . . . 9
1.12 Control volume borders in a pyramid . . . 10
1.13 Control volume borders in a hexahedron . . . 11
1.14 A face of the secondary grid . . . 11
1.15 Contributions of boundary elements to the boundary faces of point P_i . . . 12
1.16 Neighboring points of a boundary point P_i . . . 13
1.17 Queuing of hexahedra near the agglomeration front. . . . 19
1.18 Prismatic neighborhood: seed volume and direct neighbors (left), direct and indirect neighbors (right) . . . 19
1.19 Hexahedral neighborhood: direct neighbors sharing faces (left), indirect neighbors sharing edges (middle) and neighbors sharing points (right) with the seed volume . . . 19
1.20 Allowed fusing: new volume covers old volumes completely (left), new volume covers exactly one old volume (middle) and new volume lies inside the cover of one old volume . . . 20
1.21 The parts of the new volume divided by the broken line are not allowed to be fused together due to the structure of the underlying layer . . . 21
1.22 After cutting the new fusible set, some volumes are agglomerated to a previously fused cell. . . . 21
1.23 Semi coarsening with a fraction factor of 0.5 . . . 22
1.24 Coarse grid coordinates remaining in seed volume position and shifted to averaged position . . . 23
1.25 Determination of coarse grid face vectors . . . 24

1.26 Cartesian bounding boxes of tetrahedra and pyramid 25
1.27 Overlapping bounding boxes 26
1.28 Parametric space 26
1.29 The six ways to split the quadrilateral faces of the prism such that it can be subdivided into three tetrahedra 27
1.30 First configuration with no diagonal that goes through vertex p_7 (a), second configuration with one diagonal that goes through vertex p_7 (b), third configuration with two diagonals that go through vertex p_7 (c) and fourth configuration with three diagonals that go through vertex p_7 (d) 28
1.31 The two ways to split the quadrilateral face of the pyramid and to subdivide the pyramid into two tetrahedra 28
1.32 Partitioned overset unstructured grids 30
1.33 Input lists for Chimera search 30
1.34 Overset grid system and partitioned grid 30
1.35 A global ADT 30

2.1 Face for an upwind flux computation 35
2.2 Control volumes around neighboring points $P(j1)$ and $P(j2)$ 39
2.3 Control volume of point $P(j1)$ with neighboring points 42
2.4 Surrounding points used for the least square algorithm 43
2.5 Face with neighboring faces on each side 45
2.6 Face with one neighboring face on the right side 46
2.7 Face with one neighboring face on the left side 47
2.8 Flux computation for a central scheme over face F_i 50
2.9 Extrapolation of flow quantities over the boundary 53
2.10 Face of a three dimensional control volume 66

3.1 Situation at farfield boundary 76
3.2 Computation of pressure forces 81

4.1 Attached flow over both convex and concave surfaces. 98

5.1 Flow over a multi-element airfoil at angle of attack $\alpha = 21.4°$ and inflow Mach number 0.22 111
5.2 NACA0012 Euler grid 127
5.3 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers without preconditioning 127
5.4 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pold} 128
5.5 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pnew} 128
5.6 NACA0012 airfoil, angle of attack 2.0°, comparison of the residuals and C-drag for Mach number 0.001 with preconditioning 128

5.7 NACA0012 airfoil, angle of attack 2.0°, cp for Mach numbers 0.01 and 0.001 without preconditioning, with preconditioning Γ^{Pold} and preconditioning Γ^{Pnew} 129
5.8 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers without preconditioning 129
5.9 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pold} 130
5.10 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers with preconditioning Γ^{Pnew} 130
5.11 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers and LUSGS-scheme without preconditioning 130
5.12 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers and LUSGS-scheme with preconditioning Γ^{Pold} 131
5.13 NACA0012 airfoil, angle of attack 2.0°, convergence of the residuals and C-drag for different Mach numbers and LUSGS-scheme with preconditioning Γ^{Pnew} 131

6.1 Triangulation, sub-divided into two subdomains 135

9.1 $(2:4)$ refinement case 163
9.2 $(3.2:4.3)$ refinement case 166

10.1 CAD model with single oriented panel. 174
10.2 Grid topology of actuator disk periodic boundaries. 175
10.3 Actuator disks with inner hole and intersecting with wall. 175
10.4 Dualgrid cells of a cut section through the disk. 176
10.5 Partitioning of actuator disk periodic boundary. 176
10.6 Geometry data of periodic types. 176
10.7 Partitioned grid with two periodic boundaries. 177
10.8 NetCDF format of periodic boundaries of domain 0. 178
10.9 NetCDF format of periodic boundaries of domain 1. 179

11.1 Primgrid pointdata arrays of partitioned grids 181
11.2 Sequence of global ids corresponding to block membership 182

12.1 *(a)* Nomenclature of variables on a tetrahedral element *i*. *(b)* Sketch of the splitting method for pyramids. 185
12.2 Examples for edge swapping for 2d *(a)* and 3d *(b)* meshes. 187
12.3 Principle of equatorial edge swapping for $N_T = 4$. *(a)* Three-dimensional view. *(b)* View of the projected pseudo plane spanned by the points $2-5$. 188
12.4 Schematic plot of a face swapping step. 189
12.5 The edge collapsing degenerates the edge $E_{ij} = \overline{P_iP_j}$ and its surrounding elements. 189
12.6 Sketch of the movement of a node in two *(a)* and three *(b)* dimensions. 190
12.7 Equatorial edge swapping for $N_t = 3$, $N_t = 4$ and $N_t = 5$. 192
12.8 Equatorial edge swapping for $N_t = 6$ and $N_t = 7$. 193
12.9 Equatorial edge swapping for $N_t = 8$. 194

List of Tables

1.1 All possible subdivision of the hexahedrons into six tetrahedrons. 27