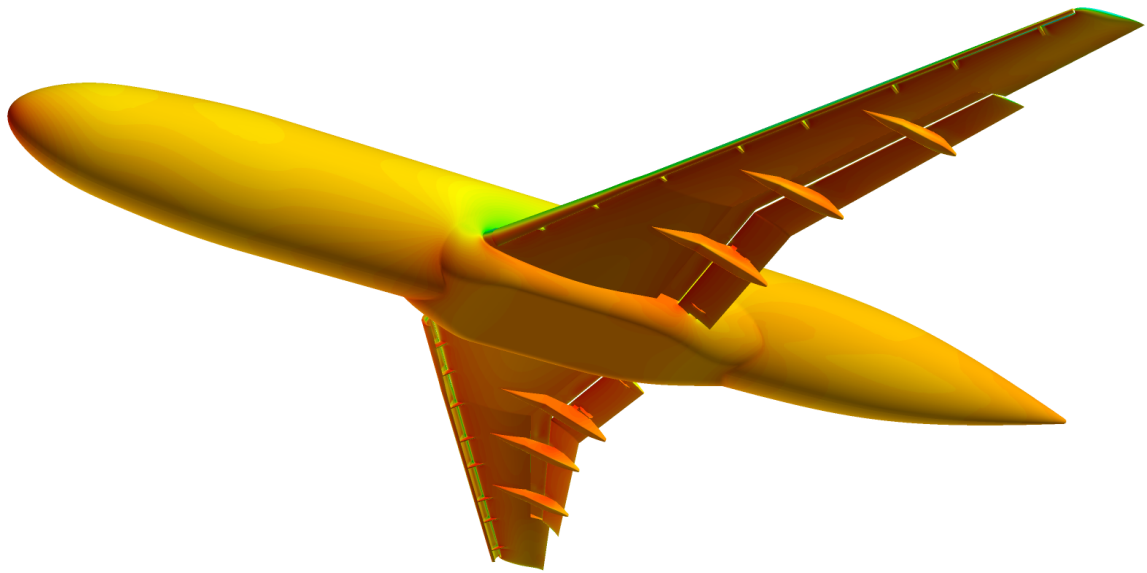


THE ARION UNSTRUCTURED GRIDS CFD SOLVER:
THEORETICAL AND USER'S MANUAL

Version 2.0 (Revision 700-349)
ISCFDC Report 2021-09



Prepared by
Ya'eer Kidron, Mark Wasserman, Yair Mor-Yossef, and Yuval Levy



COPYRIGHT © September 2021
by
The authors and the Israeli CFD Center

Copyright and Trademark Information

© 2021 ISCFDC LTD. All rights reserved. Unauthorized use, distribution or duplication is prohibited. See the legal information in the product license for the complete Legal Notice for ISCFDC proprietary software. If you are unable to access the Legal Notice, please contact ISCFDC.

Third-Party Software

This software makes use of the **PETSc** open-source library.

The following is legal information pertaining to the use and redistribution of PETSc:

Copyright (c) 1991-2021, UChicago Argonne, LLC and the PETSc Development Team
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS

OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



Contents

Abstract	xiv
1 Introduction	1
1.1 Current Release	1
1.2 Users's Manual Arrangement	1
2 Physical Models	3
2.1 Introduction	3
2.2 Single-component Perfect Gas (SPG)	3
2.2.1 Governing Equations	3
2.2.2 Integral Form For Moving Grids	5
2.2.3 Equation of State	6
2.2.4 Transport Properties	7
2.3 Gas Selection	8
3 Turbulence Models	9
3.1 RANS Turbulence Model Equations	10
3.2 The k - ω -TNT Turbulence Model	12
3.3 The k - ω -SST Model	13
3.3.1 k - ω -SST-2003 Model	14
3.4 Boundary Conditions for the k - ω -TNT Turbulence model	16
3.5 Evaluation of the Reynolds Stress Tensor	16
3.6 Turbulence-Mean-flow Coupling	17
4 Computational Methods	18

4.1	Spatial Discretization	18
4.2	Flux Approximation Schemes	19
4.2.1	HLLC	19
4.2.2	AUSM	22
4.2.2.1	AUSM ⁺ -up	23
4.2.3	Passive Scalar Approach	25
4.2.4	High Order Flux Approximations	26
4.3	Diffusive Flux Vector Discretization	27
4.4	Time Marching Schemes	27
4.4.1	Explicit schemes	27
4.4.1.1	Explicit Euler Scheme	27
4.4.1.2	Runge-Kutta Schemes	28
4.4.2	Implicit Time Marching Formulation	28
4.4.2.1	Point Gauss-Seidel	29
4.4.2.2	Runge-Kutta Schemes	29
5	Boundary Conditions	30
5.1	Introduction	30
5.2	Wall Boundary Conditions	30
5.2.1	Impermeable Wall Conditions	30
5.2.2	No-Slip Condition	31
5.2.3	Adiabatic Wall	31
5.3	Far Field Conditions	31
5.3.1	Turkel Type Conditions	31
5.3.1.1	Turkel Inlet	32
5.3.1.2	Turkel Outlet	32
5.3.2	Riemann Type Conditions	32
5.3.2.1	Riemann Inlet	32
5.3.2.2	Riemann Outlet	33
5.3.3	Fixed (Supersonic Inlet)	33
5.3.4	Extrapolation (Supersonic Outlet)	33

5.3.5	Inlet	33
5.3.6	Outlet	34
5.3.7	Inout	34
5.4	Symmetry Boundary Conditions	34
6	Computational Mesh	35
7	Parallelization	36
8	Input File, Run Preparation, and Execution	37
8.1	Preface	37
8.1.1	Scripting Language	38
8.1.2	Basic Input File: Example	39
8.2	Star-CD Grid Files	41
8.3	Conversion	42
8.3.1	Conversion Between Different Orderings	45
8.4	Log Control Options	46
8.5	Plot Control Options	49
8.6	Table Control Options	52
8.7	Parallel Options	55
8.8	Equation of State Options	57
8.9	Molecular	60
8.10	BC Options	64
8.10.1	Wall Distance Calculations	68
8.10.2	Freestream Conditions Override	69
8.11	System Options	73
8.12	Solve Options	88
8.13	PETSc Toolkit	106
8.14	UNS Options	106
8.15	Run-time Options	110
8.15.1	Run-time Options Examples	110

Bibliography

113



List of Figures

4.1	Wave structure of the HLLC approximate Riemann solver	20
-----	---	----



List of Tables

2.1	Default coefficients for Sutherland formulae (type 1)	8
2.2	Default coefficients for Sutherland formulae (type 2)	8
3.1	SST coefficients	15
8.1	Conversion	42
8.2	Metis options	43
8.2	Metis options (continued)	44
8.3	Conversion between different ordering (cnv2cnv) option	45
8.4	Log name option	46
8.5	Log prefix option	47
8.6	Log per option	47
8.7	Log stdout option	48
8.8	Log stderr option	48
8.9	Plot name option	49
8.10	Plot prefix option	49
8.11	Plot interval option	50
8.12	Plot sequential/overwrite option	50
8.13	Plot fvuns/eppic option	51
8.14	Table name option	52
8.15	Table prefix option	52
8.16	Table interval option	53
8.17	Table sequential/overwrite option	53
8.18	Table horizontal/vertical option	53
8.19	Table ray option	54

8.20	Table plane.bc option	54
8.21	Parallel cache option	55
8.22	Parallel rank option	56
8.23	equation.of.state name option	57
8.24	equation.of.state type option	58
8.25	equation.of.state R option	58
8.26	equation.of.state gamma option	58
8.27	equation.of.state ref option	59
8.28	Molecular name option	60
8.29	molecular type option	60
8.30	Molecular Mu1 option	61
8.31	Molecular Mu2 option	61
8.32	Molecular Ka1 option	62
8.33	Molecular Ka2 option	62
8.34	Molecular Eta0 option	62
8.35	Molecular C0 option	63
8.36	Molecular T0 option	63
8.37	Molecular Prandtl option	63
8.38	BC name option	64
8.39	BC type option	64
8.40	Boundary condition types	65
8.41	Boundary conditions log option	66
8.42	BC log options	67
8.43	BC wall.distance option	68
8.44	BC no.wall.distance option	68
8.45	BC Mach option	69
8.46	BC velocity.magnitude option	69
8.47	BC alpha option	70
8.48	BC beta option	70
8.49	BC u option	70
8.50	BC v option	71

8.51	BC w option	71
8.52	BC T option	71
8.53	BC p option	72
8.54	BC trb.intensity option	72
8.55	System type option	73
8.56	System types	74
8.57	System cell.gradient option	74
8.58	Cell gradient types	75
8.59	System face.gradient option	75
8.60	Face gradient types	76
8.61	Limiter option	76
8.62	Limiter types	77
8.63	System venka.k option	77
8.64	System CFL option	78
8.65	System implicit.jacobi option	79
8.66	System convection.noslip.diagonal option	80
8.67	System convection.impermeable.diagonal option	80
8.68	System convection.symmetry.diagonal option	81
8.69	System realizability.trb.w option	81
8.70	System source.mpk option	82
8.71	System damp option	82
8.72	System relaxation option	83
8.73	System iteration.convergence option	83
8.74	System log option	84
8.75	System log options	85
8.76	System plot option	86
8.77	Plot functions	87
8.78	Solve convection.flux option	88
8.79	Convection flux types	88
8.80	Solve convection.jacobian option	89
8.81	Convection Jacobian types	89

8.82	Solve spatial.order option	90
8.83	Solve sweeps option	90
8.84	Solve time.march option	91
8.85	Time marching methods	92
8.86	Solve iterations option	93
8.87	Solve time.step.reduction option	93
8.88	Solve dual.time option	94
8.89	Solve global.minimal.timestep option	94
8.90	Solve save option	94
8.91	Solve save.path option	95
8.92	Solve save.sequential option	95
8.93	Solve load.path option	95
8.94	Solve load.sequence option	96
8.95	Solve eos option	96
8.96	Solve molecular option	96
8.97	Solve p option	97
8.98	Solve T option	97
8.99	Solve Mach option	97
8.100	Solve velocity magnitude option	98
8.101	Solve u option	98
8.102	Solve v option	98
8.103	Solve w option	99
8.104	Solve alpha option	99
8.105	Solve beta option	99
8.106	Solve trb.intensity option	100
8.107	Solve trb.mt option	100
8.108	Solve reference.mach.trb option	101
8.109	Solve reference.velocity.trb option	101
8.110	Solve reference.mach option	101
8.111	Solve reference.velocity option	102
8.112	Solve reference.length option	102

8.113	Solve ref option	103
8.114	Solve plot option	103
8.115	Solve plot options	104
8.116	Solve log option	104
8.117	Solve log options	105
8.118	UNS name option	106
8.119	UNS prefix option	107
8.120	UNS key option	107
8.121	UNS scale option	108
8.122	UNS log option	108
8.123	UNS log options	109
8.124	Run-time options	110
8.125	Run-time options examples	111

Listings

8.1 Example of a simple Arion input file	39
---	----



Abstract

This manual describes the algorithms, methods, and input and output files of the **Arion** code. In addition, the manual serves as the user's manual of the code. The current version of the code is the first release production version of the code, still undergoing continuous development efforts. As of the current revision, the code provides the capability to simulate inviscid, laminar, and turbulent steady flows. Currently, the flow solver contains the HLLC approximate Riemann solver, the AUSM⁺-up, or the AUFSR+ scheme for the approximation of the convective fluxes and the k- ω -TNT turbulence model. The code is fully parallel using a distributed architecture, as well as by shared memory (OpenMP) using hybrid architecture.



Chapter 1

Introduction

This manual describes the algorithms, methods, and input/output files of the **Arion** code. In addition, the manual serves as the user's manual of the code.

1.1 Current Release

The current version of the code has gone through various changes. As of the current revision, the code provides the capability to simulate inviscid, laminar, and turbulent steady flows. Currently, the flow solver contains the HLLC approximate Riemann solver, the AUSM⁺-up scheme, or the AUFSR+ scheme for the approximation of the convective fluxes. Turbulence may be modeled using either the $k-\omega$ -TNT or the $k-\omega$ -SST turbulence models. The code is fully parallel using a distributed architecture. Domain decomposition is carried out using the Metis open source library.

1.2 Users's Manual Arrangement

The report is arranged in the following manner: Chapter 2 contains a description of the available physical models while Chapter 3 describes the turbulence models that are used in the code. Chapter 4 is dedicated to a detailed description of the computational methods. Chapter 5 briefly describes the boundary conditions while Chapter 6 describes the type and format of unstructured meshes that the code sup-



ports. Chapter 7 describes the parallelization of the code. Chapter 8 contains a detailed description of the input file syntax, and actually serves as the code's reference manual.



Chapter 2

Physical Models

2.1 Introduction

Computer simulations are generally based upon the numerical solution of the model equations in a discretized mode. The accuracy of the computations depends mainly on the physical modeling, the numerical algorithm, and the quality of the computational mesh. At its current developmental stage, the **Arion** code provides the capability to simulate only single-component perfect gas flows, in particular, air flows. This chapter contains a description of the physical models that are available in the solver.

2.2 Single-component Perfect Gas (SPG)

2.2.1 Governing Equations

The equations governing single-component, perfect gas fluid flow are derived from the laws of conservation of mass, momentum, and total energy. The set of five partial differential equations is known as the Navier-Stokes equations and can be represented in a conservation-law form that is convenient for numerical simulations, namely

$$\frac{\partial Q}{\partial t} + \frac{\partial (E_c - E_d)}{\partial x} + \frac{\partial (F_c - F_d)}{\partial y} + \frac{\partial (G_c - G_d)}{\partial z} = 0 \quad (2.1)$$

where Q is the vector of conserved mass, momentum, and energy

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \quad (2.2)$$

where the density is denoted by ρ , the Cartesian velocity vector components are denoted by u, v and w , and E denotes the total (internal and kinetic) energy of the gas. The inviscid flux vectors, E_c , F_c , and G_c , are

$$E_c = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E + p) \end{bmatrix}, \quad F_c = \begin{bmatrix} \rho v \\ \rho v^2 + p \\ \rho vw \\ \rho vw \\ v(E + p) \end{bmatrix}, \quad G_c = \begin{bmatrix} \rho w \\ \rho w^2 + p \\ \rho vw \\ \rho w^2 + p \\ w(E + p) \end{bmatrix} \quad (2.3)$$

and the viscous flux vectors, E_d , F_d , and G_d , are

$$E_d = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \beta_x \end{bmatrix}, \quad F_d = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \beta_y \end{bmatrix}, \quad G_d = \begin{bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \end{bmatrix} \quad (2.4)$$

where

$$\begin{aligned}
\tau_{xx} &= \lambda \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right) + 2\mu \frac{\partial u}{\partial x} \\
\tau_{yy} &= \lambda \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right) + 2\mu \frac{\partial v}{\partial y} \\
\tau_{zz} &= \lambda \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right) + 2\mu \frac{\partial w}{\partial z} \\
\tau_{xy} &= \tau_{yx} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \\
\tau_{xz} &= \tau_{zx} = \mu \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \right) \\
\tau_{yz} &= \tau_{zy} = \mu \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right) \\
\beta_x &= u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + \kappa \frac{\partial T}{\partial x} \\
\beta_y &= u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + \kappa \frac{\partial T}{\partial y} \\
\beta_z &= u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + \kappa \frac{\partial T}{\partial z}
\end{aligned} \tag{2.5}$$

where T is the temperature. Stokes hypothesis, $\lambda = -\frac{2}{3}\mu$, is typically used to further simplify Equation (2.5).

Finally, in the perfect-gas model, the source-term vector, S , may only contain contributions from the turbulence model (see Chapter 3).

2.2.2 Integral Form For Moving Grids

For a three-dimensional flow through a finite volume Ω , enclosed by the boundary surface $\partial\Omega \equiv S$ that is moving with a grid velocity \bar{V}_g , the integral form of the conservation equations in an inertial frame of reference is given as:

$$\frac{\partial}{\partial t} \int_{\Omega} Q d\Omega + \oint_S d\bar{S} \cdot \bar{H} = 0 \tag{2.6}$$

where

$$\bar{H} = (E_c - E_d) \hat{i} + (F_c - F_d) \hat{j} + (G_c - G_d) \hat{k} \tag{2.7}$$

In Equation (2.6) the vector of dependent variables, Q , remains as in Equation (2.2). Similarly, the viscous flux vectors, E_d , F_d , and G_d , appearing in Equation (2.7), remain as in Equations (2.4) and (2.5) (see Section 2.2.1). In contrast, the inviscid flux vectors



take a form that reflects the motion of the grid as follows:

$$\begin{aligned}
 E_c &= \begin{bmatrix} \rho(u - u_g) \\ \rho u(u - u_g) + p \\ \rho v(u - u_g) \\ \rho w(u - u_g) \\ (E + p)(u - u_g) + u_g p \end{bmatrix} \\
 F_c &= \begin{bmatrix} \rho(v - v_g) \\ \rho u(v - v_g) \\ \rho v(v - v_g) + p \\ \rho w(v - v_g) \\ (E + p)(v - v_g) + v_g p \end{bmatrix} \\
 G_c &= \begin{bmatrix} \rho(w - w_g) \\ \rho u(w - w_g) \\ \rho v(w - w_g) \\ \rho w(w - w_g) + p \\ (E + p)(w - w_g) + w_g p \end{bmatrix}
 \end{aligned} \tag{2.8}$$

where u_g , v_g , and w_g are the Cartesian components of the grid velocity vector \bar{V}_g . Note that currently the grid velocity vector is set to $\bar{V}_g \equiv 0$.

2.2.3 Equation of State

To close the system of fluid dynamics equations, it is necessary to establish relations between the thermodynamics variables, p , ρ , T , and the internal energy, e_I . Assuming a perfect gas, the pressure and temperature may be obtained from the following equation of state:

$$p = \rho R T \tag{2.9}$$

where R is the gas constant ($R = 287.0$ for air). By assuming further that the gas is a calorically perfect gas (and hence the specific heats C_p and C_v are constant), the

equation of state takes the form:

$$p = \rho(\gamma - 1)e_I \quad (2.10)$$

where e_I is the internal energy of the gas, and γ is the (constant) ratio of specific heats (c_p/c_v). In terms of the flow variables, the pressure and temperature are calculated using:

$$\begin{aligned} p &= (\gamma - 1) \left[E - \frac{1}{2} \rho (u^2 + v^2 + w^2) \right] \\ T &= \frac{\gamma - 1}{R} \left[e - \frac{1}{2} (u^2 + v^2 + w^2) \right] \end{aligned} \quad (2.11)$$

where $e = \frac{E}{\rho}$ is the specific total energy.

2.2.4 Transport Properties

In addition to the equation of state, it is also necessary to establish constitutive relations, namely, relations for the coefficient of viscosity, μ , and the coefficient of thermal conductivity, κ . In the single-component perfect gas (SPG) model, the Sutherland formulae are exclusively used to evaluate these coefficients. The **Arion** code provides two different ways to evaluate μ and κ .

The first (called “molecular.sutherland.air.1”, see Table 8.29) is based on the following relations:

$$\begin{aligned} \mu &= C_{\mu_1} \frac{T^{\frac{3}{2}}}{T + C_{\mu_2}} \\ \kappa &= C_{\kappa_1} \frac{T^{\frac{3}{2}}}{T + C_{\kappa_2}} \end{aligned} \quad (2.12)$$

The default of the coefficients C_{μ_1} , C_{μ_2} , C_{κ_1} , and C_{κ_2} correspond to air and are given in Table 2.1. For the purpose of simulating a gas whose transport properties are different than air, the coefficients can be set by the user using the directives as described in Tables 8.30, 8.31, 8.32, 8.33.

Fluid type	C_{μ_1}	C_{μ_2}	C_{κ_1}	C_{κ_2}
Air	1.458×10^{-6}	110.4	2.495×10^{-3}	194

Table 2.1: Default coefficients for Sutherland formulae (type 1)

The second way to evaluate μ and κ (called “molecular.sutherland.air.2”, see Table 8.29) is based on the following relations:

$$\begin{aligned}\mu &= \eta_0 \frac{T_0 - C_0}{T - C_0} \left(\frac{T}{T_0} \right)^{\frac{3}{2}} \\ \kappa &= \frac{c_p \mu}{Pr}\end{aligned}\tag{2.13}$$

where Pr is the Prandtl number. The default of the coefficients η_0 , T_0 , and C_0 correspond to air and are given in Table 2.2. The coefficients can be set by the user using the directives as described in Tables 8.34, 8.35, 8.36, and 8.37.

Fluid type	η_0	C_0	T_0	Pr
Air	1.827×10^{-5}	120	291.15	0.72

Table 2.2: Default coefficients for Sutherland formulae (type 2)

2.3 Gas Selection

As mentioned above, the **Arion** code provides the means to simulate the flow of any perfect gas. This is facilitated via a series of directives that allow to set the specific gas constant, (R , see Table 8.25), the the heat capacity ratio, (γ , see Table 8.26, and the Sutherland formulae coefficients as described in Section 2.2.4.



Chapter 3

Turbulence Models

The unsteady Navier-Stokes equations are generally considered to govern turbulent flows in the continuum flow regime. However, turbulent flow cannot be numerically simulated as easily as laminar flow. To resolve a turbulent flow by direct numerical simulation (DNS) requires that all relevant length scales be properly resolved. Such requirements place great demands on the computer resources, a fact that renders the possibility of conducting DNS analysis about complete aircraft configurations infeasible.

A practical approach to simulating turbulent flows is to solve the time-averaged Navier-Stokes equations. These equations are known as the “Reynolds averaged Navier-Stokes” (RANS) equations. The averaging of the equations of motion gives rise to new terms that are called the Reynolds stresses. To solve the averaged equations, the Reynolds stress tensor must be related to the flow variables through turbulence models. The models are used to “close” the system through an additional set of assumptions. The models are classified based on the number of additional partial differential equations that must be solved. The **Arion** code currently contains only one turbulence model, a two-equation model.



3.1 RANS Turbulence Model Equations

The **Arion** code treats the mean flow and turbulence models equations in a unified manner. To this end, the Navier-Stokes equation set is extended to include the turbulence model equations. Consequently, the discretization of the various fluxes can be conducted in the same manner.

The equations governing turbulent flows are obtained by Favre-averaging the Navier-Stokes equations and by modeling the Reynolds stress tensor. The unknown averaged Reynolds stress tensor is modeled either using the Boussinesq assumption via a linear eddy-viscosity model or by directly solving a transport equation for each of the Reynolds stress components via a second moment closure. The general form of the resulting Navier-Stokes equations and the turbulence model equations has the form (for simplicity of the representation, with no loss of generality, the formulation under the perfect gas physical model assumption is brought herein; it can be easily extended to any physical model):

$$\frac{\partial Q}{\partial t} + \frac{\partial (E_c - E_d)}{\partial x} + \frac{\partial (F_c - F_d)}{\partial y} + \frac{\partial (G_c - G_d)}{\partial z} = S \quad (3.1)$$

where S is the source term associated with the turbulence model only (once again, under the perfect gas physical model assumption). Hence, for turbulent flow simulations, Equation (3.1) replaces Equation (2.1).

In what follows, the symbol $(\bar{\quad})$ indicates non-weighted averaging, the symbol $(\tilde{\quad})$ signifies mass weighted Favre averaging, and the symbol $(\prime\prime)$ denotes Favre fluctuations. Depending on whether the turbulence model has one, two, or m equations, the vector of dependent variables, Q , and the vector of source terms S now take the



form:

$$Q = \begin{bmatrix} \bar{\rho} \\ \bar{\rho}\tilde{u} \\ \bar{\rho}\tilde{v} \\ \bar{\rho}\tilde{w} \\ \tilde{E} \\ \bar{\rho}q_1 \\ \dots \\ \bar{\rho}q_m \end{bmatrix}, \quad S = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ s_1 \\ \dots \\ s_m \end{bmatrix} \quad (3.2)$$

where q is the vector of turbulence model dependent variables and s , is the source terms vector. Note that the source terms differ from model to model. The vectors E_c , F_c , and G_c usually take the form:

$$E_c = \begin{bmatrix} \bar{\rho}\tilde{u} \\ \bar{\rho}\tilde{u}^2 + \bar{p} \\ \bar{\rho}\tilde{u}\tilde{v} \\ \bar{\rho}\tilde{u}\tilde{w} \\ \tilde{u}(\tilde{E} + \bar{p}) \\ \bar{\rho}\tilde{u}q_1 \\ \dots \\ \bar{\rho}\tilde{u}q_m \end{bmatrix}, \quad F_c = \begin{bmatrix} \bar{\rho}\tilde{v} \\ \bar{\rho}\tilde{u}\tilde{v} \\ \bar{\rho}\tilde{v}^2 + \bar{p} \\ \bar{\rho}\tilde{v}\tilde{w} \\ \tilde{v}(\tilde{E} + \bar{p}) \\ \bar{\rho}\tilde{v}q_1 \\ \dots \\ \bar{\rho}\tilde{v}q_m \end{bmatrix}, \quad G_c = \begin{bmatrix} \bar{\rho}\tilde{w} \\ \bar{\rho}\tilde{u}\tilde{w} \\ \bar{\rho}\tilde{v}\tilde{w} \\ \bar{\rho}\tilde{w}^2 + \bar{p} \\ \tilde{w}(\tilde{E} + \bar{p}) \\ \bar{\rho}\tilde{w}q_1 \\ \dots \\ \bar{\rho}\tilde{w}q_m \end{bmatrix} \quad (3.3)$$

Similarly, the vectors, E_d , F_d , and G_d , usually take the form:

$$E_d = \begin{bmatrix} 0 \\ \bar{\tau}_{xx} - \overline{\rho u'' u''} \\ \bar{\tau}_{yx} - \overline{\rho v'' u''} \\ \bar{\tau}_{zx} - \overline{\rho w'' u''} \\ \bar{\beta}_x \\ e_{d_1} \\ \dots \\ e_{d_m} \end{bmatrix}, \quad F_d = \begin{bmatrix} 0 \\ \bar{\tau}_{xy} - \overline{\rho u'' v''} \\ \bar{\tau}_{yy} - \overline{\rho v'' v''} \\ \bar{\tau}_{zy} - \overline{\rho w'' v''} \\ \bar{\beta}_y \\ f_{d_1} \\ \dots \\ f_{d_m} \end{bmatrix}, \quad G_d = \begin{bmatrix} 0 \\ \bar{\tau}_{xz} - \overline{\rho u'' w''} \\ \bar{\tau}_{yz} - \overline{\rho v'' w''} \\ \bar{\tau}_{zz} - \overline{\rho w'' w''} \\ \bar{\beta}_z \\ g_{d_1} \\ \dots \\ g_{d_m} \end{bmatrix} \quad (3.4)$$

where the vectors e_d , f_d , and g_d differ from model to model and

$$\begin{aligned}
 \bar{\beta}_x &= \tilde{u} (\bar{\tau}_{xx} - \overline{\rho u'' u''}) + \tilde{v} (\bar{\tau}_{xy} - \overline{\rho u'' v''}) + \tilde{w} (\bar{\tau}_{xz} - \overline{\rho u'' w''}) + (\bar{\kappa} + \bar{\kappa}_t) \frac{\partial \bar{T}}{\partial x} \\
 \bar{\beta}_y &= \tilde{u} (\bar{\tau}_{yx} - \overline{\rho v'' u''}) + \tilde{v} (\bar{\tau}_{yy} - \overline{\rho v'' v''}) + \tilde{w} (\bar{\tau}_{yz} - \overline{\rho v'' w''}) + (\bar{\kappa} + \bar{\kappa}_t) \frac{\partial \bar{T}}{\partial y} \\
 \bar{\beta}_z &= \tilde{u} (\bar{\tau}_{zx} - \overline{\rho w'' u''}) + \tilde{v} (\bar{\tau}_{zy} - \overline{\rho w'' v''}) + \tilde{w} (\bar{\tau}_{zz} - \overline{\rho w'' w''}) + (\bar{\kappa} + \bar{\kappa}_t) \frac{\partial \bar{T}}{\partial z}
 \end{aligned} \tag{3.5}$$

The terms $\bar{\kappa}$ and $\bar{\kappa}_t$ are the averaged molecular and turbulent heat conductivities, respectively. The molecular heat conductivity is calculated using Sutherland's law (see Equation (2.12)) while the turbulent heat conductivity is calculated using

$$\bar{\kappa}_t = \frac{c_p \bar{\mu}_t}{Pr_t} \tag{3.6}$$

where $\bar{\mu}_t$ denotes the turbulent viscosity. The term c_p is the specific heat capacity at constant pressure, Pr is the Prandtl number set to $Pr = 0.72$, and Pr_t is the turbulent Prandtl number set to $Pr_t = 0.9$. The average turbulent viscosity, $\bar{\mu}_t$ differs from model to model.

3.2 The k- ω -TNT Turbulence Model

The TNT turbulence model has two clear advantages over other two-equation turbulence models: it uses a topology-free approach, and it is insensitive to the specific turbulence dissipation rate free-stream boundary condition. The source term of the model is given by:

$$S = \left\{ \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ P_k - \beta_k \bar{\rho} k \omega \\ \alpha_\omega \frac{\omega}{k} P_k - \beta_\omega \bar{\rho} \omega^2 + \max(\mathcal{E}, 0) \end{array} \right\} \tag{3.7}$$



The production term is denoted by P_k and it is based on the Boussinesq approximation while \mathcal{E} is the cross diffusion term. The turbulent viscosity is defined as

$$\mu_t = \frac{\bar{\rho}k}{\omega} \quad (3.8)$$

The remaining model constants are $\sigma_k = 1.5$, $\sigma_\omega = 2.0$, $\sigma_d = 0.5$, $\beta_\omega = 0.075$, $\beta_k = 0.09$, $\alpha_\omega = \frac{\beta}{\beta^*} - \frac{\sigma_\omega \kappa^2}{\sqrt{\beta^*}}$, with $\kappa = 0.41$.

A description of the boundary conditions appears in Section 3.4.

3.3 The k - ω -SST Model

There are numerous suggestions in the literature for two-equation turbulence models, many being variations of a few baseline models. As most two equation models contain one equation for the turbulence kinetic energy, k , one important issue is the quantity chosen to represent the length scale. While ϵ is the most obvious and popular option, it is one that causes significant problems in practice, especially in near-wall flows approaching separation. In computational aerodynamics, the most popular alternative to ϵ itself is the turbulent specific dissipation rate, ω . The attraction of ω -based models is rooted in the observation that it gives a superior representation of the near-wall behavior, especially in adverse pressure gradient regions. On the other hand, one serious flaw exhibited by ω based models is the extreme sensitivity to the value of ω at irrotational boundaries of shear flows and, by implication, also to the value in weak-shear regions within a complex shear flow. This, as well as other defects, have led Menter [1] to formulate a hybrid model which blends the k - ω model near-wall regions with the k - ϵ model in regions that are far from walls. In recent years, this model has become the most popular two-equation model in aeronautical CFD practice, especially in weakly separated flows. Again, over the years several variations and modifications to the original k - ω -SST model have appeared. The k - ω -SST-2003 (following the naming convention from the [TMR website](#)) has been chosen and implemented in the EZAir suite.

3.3.1 k - ω -SST-2003 Model

The transport form of the compressible k - ω -SST-2003 turbulence model contains two transport equations. A transport equation for the turbulence kinetic energy, k , and a second transport equation for the turbulent specific dissipation rate, ω . These transport equations take the following form:

$$\frac{\partial \rho k}{\partial t} + \frac{\partial \rho u_j k}{\partial x_j} = \frac{\partial}{\partial x_j} \left[(\mu + \sigma_k \mu_t) \frac{\partial k}{\partial x_j} \right] + P_k - \beta^* \rho \omega k \quad (3.9)$$

$$\begin{aligned} \frac{\partial \rho \omega}{\partial t} + \frac{\partial \rho u_j \omega}{\partial x_j} &= \frac{\partial}{\partial x_j} \left[(\mu + \sigma_\omega \mu_t) \frac{\partial \omega}{\partial x_j} \right] + \frac{\gamma_\omega}{\mu_t} \rho P_k - \beta \rho \omega^2 \\ &+ 2(1 - F_1) \frac{\rho \sigma_\omega}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} \end{aligned} \quad (3.10)$$

where t denotes the time and $x_j=[x,y,z]$ denote the Cartesian coordinates. The fluid density is denoted by ρ while the Cartesian velocity vector components are denoted by $u_j=[u,v,w]$. The production term, denoted by P_k , is defined as

$$P_k = \Re_{ij} \frac{\partial u_i}{\partial x_j} \quad (3.11)$$

where $\Re_{ij} = \overline{\rho u_i'' u_j''}$ are the Reynolds stress tensor components.

The coefficient β^* is a constant equal to $\beta^* = 0.09$. The rest of the model coefficients, $\phi_c=(\sigma_k, \sigma_\omega, \gamma_\omega, \beta)$ are blended according to

$$\phi_c = F_1 \phi_1 + (1 - F_1) \phi_2 \quad (3.12)$$

where the coefficients ϕ_1 and ϕ_2 are given in Table 3.1.

The F_1 function, proposed by Menter [1] himself, is given as:

$$F_1 = \tanh(\mathfrak{Z}_1^4) \quad (3.13)$$

	σ_k	σ_ω	γ_ω	β
ϕ_1	0.856	0.5	5/9	0.075
ϕ_2	1.0	0.856	0.44	0.0828

Table 3.1: SST coefficients

with the argument \mathfrak{Z}_1 given by:

$$\mathfrak{Z}_1 = \min \left[\max \left(\frac{\sqrt{k}}{\beta^* \omega d}, \frac{500\mu}{\rho \omega d^2} \right), \frac{4\sigma_{\omega 2} \rho k}{\max \left(2\sigma_{\omega 2} \frac{\rho}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j}, 10^{-10} \right) d^2} \right] \quad (3.14)$$

The turbulent viscosity is defined as

$$\mu_t = \frac{a_1 \rho k}{\max(a_1 \omega, S F_2)} \quad (3.15)$$

where S is:

$$S = \sqrt{2 S_{ij} S_{ij}} \quad (3.16)$$

with

$$S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (3.17)$$

and the function F_2 is defined as follow:

$$F_2 = \tanh(\mathfrak{Z}_2^2) \quad (3.18)$$

with

$$\mathfrak{Z}_2 = \max \left(2 \frac{\sqrt{k}}{\beta^* \omega d}, \frac{500\nu}{\omega d^2} \right) \quad (3.19)$$

In the k - ω -SST-2003 version, the production term, P_k is limited as follows

$$P_k = \min(P_k, 10\beta^* \rho \omega k) \quad (3.20)$$

A description of the boundary conditions appears in Section 3.4.



3.4 Boundary Conditions for the k- ω -TNT Turbulence model

To close the solution of any of the k- ω models, the boundary conditions should be specified. The no-slip wall boundary conditions of k and ω , denoted by k_{wall} and ω_{wall} , respectively, are specified as follows:

$$k_{wall} = 0 \quad (3.21)$$

$$\omega_{wall} = 10 \frac{6\nu}{\beta_1 (\Delta d_1)^2} \quad (3.22)$$

where Δd_1 denotes the distance between the center of the first cell neighboring the wall and the wall. The inflow boundary condition of k , denoted by k_∞ for external flows is

$$k_\infty = \frac{3}{2} (Tu \cdot U_\infty)^2 \quad (3.23)$$

where Tu represents the turbulence intensity and U_∞ is the magnitude of the inflow velocity. The inflow boundary condition of ω , denoted as ω_∞ is specified as follows:

$$\omega_\infty = \frac{\bar{\rho}_\infty k_\infty}{(\mu_t)_\infty} \quad (3.24)$$

with the recommended values of $(\mu_t)_\infty$ for external flows are as follows:

$$0.01 < \frac{(\mu_t)_\infty}{(\mu)_\infty} < 1.0 \quad (3.25)$$

3.5 Evaluation of the Reynolds Stress Tensor

The TNT model is a linear eddy viscosity model (LEVM), and therefore the Reynolds stress tensor that is added to the mean flow equations is defined based on the Boussinesq assumption, namely:

$$\mathfrak{R}_{ij} = \mathfrak{R}_{ij}^{LEVM} \quad (3.26)$$

where

$$\mathfrak{R}_{ij}^{LEVM} = 2\mu_t \left(S_{ij} - \frac{1}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right) - \frac{2}{3} \bar{\rho} k \delta_{ij} \quad (3.27)$$

and

$$S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (3.28)$$

Note that the turbulence kinetic energy k is not available with the SA-Edwards model, therefore the term $\frac{2}{3} \bar{\rho} k \delta_{ij}$ is neglected when using the SA-Edwards model.

3.6 Turbulence-Mean-flow Coupling

In eddy-viscosity-based models, the coefficients of viscosity μ and thermal conductivity κ are replaced by the relations

$$\begin{aligned} \mu &= \bar{\mu}_l + \bar{\mu}_t \\ \kappa &= \bar{\kappa}_l + \frac{C_p \bar{\mu}_t}{Pr_t} \end{aligned} \quad (3.29)$$

to account for the effects of turbulence on the mean-flow. The turbulent Prandtl number is assumed constant ($Pr_t = 0.9$).



Chapter 4

Computational Methods

4.1 Spatial Discretization

A conservative cell-centered finite volume methodology is employed to discretize the governing equations. The computational domain is a unstructured hybrid grid that is discretized into N_{cv} non-overlapping control volumes. A control volume, C_v is defined by a grid volume element and $\partial\Gamma$ is the volume control surface, with $\mathbf{n} = [n_x, n_y, n_z]^T$ being the outward-pointing, unit normal vector to $\partial\Gamma$. Therefore, Equation (2.6) for a control volume C_v can be expressed as:

$$\frac{\partial}{\partial t} \int_{C_v} Q dV + \int_{\partial\Gamma} H dS = 0 \quad (4.1)$$

where H is the rotated flux, namely,

$$\begin{aligned} H &= H_c + H_d \\ H_c &= E_c n_x + F_c n_y + G_c n_z \\ H_d &= E_d n_x + F_d n_y + G_d n_z \end{aligned} \quad (4.2)$$



The term H_c is the convective part of the flux while H_d is the diffusive part of the flux. The semi-discrete form of Equation(4.1) for a non-deforming cell i is given by:

$$V_i \frac{dQ_i}{dt} = - \sum_{j \in N(i)} H_{ij} S_{ij} = R_i \quad (4.3)$$

where V_i denotes the cell volume, Q_i is the vector of cell-averaged conservative dependent variables, $N(i)$ denotes the set of cell i neighbors, H_{ij} is the rotated flux vector normal to the interface ij shared by cell i and cell j , and S_{ij} is the interface area. The number of cell neighbors, $N(i)$, depends on the type of the cell element. For example, a tetrahedron has 4 neighbors and therefore $N(i) = 4$ whereas for a hexahedron it is $N(i) = 6$. The term R_i signifies the residual of the equations. In what follows, the subscript “ i ” is dropped for compactness of the representation.

4.2 Flux Approximation Schemes

In flux difference splitting, the problem of computing the cell-face fluxes for a control volume is viewed as a series of one-dimensional Riemann problems along the direction normal to the control-volume faces. Because some of the details of the exact solution, obtained at considerable cost, are lost in the cell-averaged representation of the data, the solution of the full Riemann problem is usually replaced by methods referred to as approximate Riemann solvers. In what follows, the currently employed HLLC scheme is described in detail.

4.2.1 HLLC

The concept of average-state approximations was introduced by Harten, Lax and van-Leer [2] in 1983. The Harten, Lax and van-Leer (HLL) scheme is attractive because of its robustness, conceptual simplicity, and ease of coding, but it has the serious flaw of a diffusive contact surface. This is mainly because the HLL solver reduces the exact Riemann problem to two pressure waves and therefore neglects the contact surface. Toro *et al* [3] discussed this limitation, and proposed a modified three wave

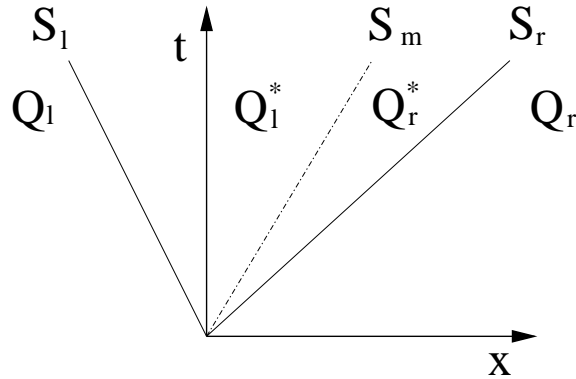


Figure 4.1: Wave structure of the HLLC approximate Riemann solver

solver, named HLLC, where the contact discontinuity is explicitly present. The HLLC scheme is found to have the following properties:

1. Exact preservation of isolated contact discontinuities and shear waves.
2. Positivity preserving of a scalar quantity.
3. Enforcement of the entropy condition.

The resulting scheme greatly improves contact discontinuity resolution and has been successfully used to compute compressible viscous and turbulent flows [4].

The HLLC approximate Riemann solver that is implemented in the **Arion** code is as proposed by Batten *et al* [4]. The HLLC scheme assumes two intermediate states, \mathbf{Q}_l^* and \mathbf{Q}_r^* within the region bounded by the left moving wave, S_l , and the right moving wave, S_r (the subscripts “*l*” and “*r*” denote the left and right states of the approximate Riemann problem, respectively). The states \mathbf{Q}_l^* and \mathbf{Q}_r^* are split by the contact discontinuity, which moves with the velocity S_m (see Figure 4.1).

Two wave speed estimates can be used. In the first, the wave speeds S_l and S_r are computed according to Einfeldt *et al* [5] as follows:

$$S_l = \min[\lambda_{min}, \lambda_{min}^{Roe}] \quad (4.4)$$

$$S_r = \max[\lambda_{max}, \lambda_{max}^{Roe}] \quad (4.5)$$

where λ_{min} is the smallest eigenvalue and λ_{max} is the largest eigenvalue, evaluated at the interface. Similarly, λ_{min}^{Roe} and λ_{max}^{Roe} are the smallest and largest eigenvalues of Roe's average matrix [6], respectively. In the second, the wave speeds S_l and S_r are computed according to Davis [7] as follows:

$$S_l = \min[\lambda_{min}(L), \lambda_{min}(R)] \quad (4.6)$$

$$S_r = \max[\lambda_{max}(L), \lambda_{max}(R)] \quad (4.7)$$

The normal velocity to the interface, denoted by q , is defined as:

$$q = (u - u_g) n_x + (v - v_g) n_y + (w - w_g) n_z \quad (4.8)$$

The contact discontinuity speed S_m is evaluated according to Batten *et al* [4] by

$$S_m = \frac{\rho_r q_r (S_r - q_r) - \rho_l q_l (S_l - q_l) + p_l - p_r}{\rho_r (S_r - q_r) - \rho_l (S_l - q_l)} \quad (4.9)$$

This choice of S_m enforces the equality of the two star pressures, *i.e.*, $p^* = p_l^* = p_r^*$ which is obtained from

$$p^* = \rho_l (q_l - S_l) (q_l - S_m) + p_l = \rho_r (q_r - S_r) (q_r - S_m) + p_r \quad (4.10)$$

Introducing the intermediate left state vector

$$\mathbf{Q}_l^* = \begin{Bmatrix} \rho_l^* \\ (\rho u)_l^* \\ (\rho v)_l^* \\ (\rho w)_l^* \\ E_l^* \end{Bmatrix} = \Omega_l \begin{Bmatrix} \rho_l (S_l - q_l) \\ (S_l - q_l) (\rho u)_l + (p^* - p_l) n_x \\ (S_l - q_l) (\rho v)_l + (p^* - p_l) n_y \\ (S_l - q_l) (\rho w)_l + (p^* - p_l) n_z \\ (S_l - q_l) E_l - p_l q_l + p^* S_m \end{Bmatrix} \quad (4.11)$$



where $\Omega_l \equiv (S_l - S_m)^{-1}$. The left state flux vector becomes

$$\mathbf{H}_{c_l}^* \equiv \mathbf{H}_c(\mathbf{Q}_l^*) = \mathbf{Q}_l^* S_m + \begin{pmatrix} 0 \\ p^* n_x \\ p^* n_y \\ p^* n_z \\ p^* S_m \end{pmatrix} \quad (4.12)$$

and the corresponding intermediate right state vector and right flux vector are obtained from Equations (4.11), (4.12) by interchanging the subscripts $l \rightarrow r$. Finally, the numerical HLLC flux is defined as follow

$$\mathbf{H}_c(\mathbf{Q}_l, \mathbf{Q}_r) = \begin{cases} \mathbf{H}_c(\mathbf{Q}_l) & \text{if } S_l > 0 \\ \mathbf{H}_c(\mathbf{Q}_l^*) & \text{if } S_l \leq 0 < S_m \\ \mathbf{H}_c(\mathbf{Q}_r^*) & \text{if } S_m \leq 0 \leq S_r \\ \mathbf{H}_c(\mathbf{Q}_r) & \text{if } S_r < 0 \end{cases} \quad (4.13)$$

where $\mathbf{H}_c(\mathbf{Q}_l)$ and $\mathbf{H}_c(\mathbf{Q}_r)$ are the left and right analytic flux vectors, respectively.

4.2.2 AUSM

The advection upstream splitting method (AUSM) was first introduced in the year 1993 by Liou and Steffen [8]. The development of the AUSM was motivated by the desire to combine the efficiency of flux vector splitting methods (FVS) and the accuracy of flux differencing splitting methods (FDS). The key idea behind AUSM schemes is the the fact that the inviscid flux vector consists of two physically distinct parts, namely the *convective* terms and the *pressure* terms. The convective terms can therefore be considered as passive scalar quantities convected by a suitably defined velocity. On the other hand, the pressure flux terms are governed by the acoustics wave speeds.



4.2.2.1 AUSM⁺-up

Although AUSM schemes enjoy a demonstrated improvement in accuracy, efficiency, and robustness over existing schemes, they have been found to have deficiencies in some cases. In the year 1996, Liou improved the original AUSM, termed now the AUSM⁺ [9]. Among the improvement features of the original AUSM scheme are the following properties: (1) exact resolution of a one-dimensional contact discontinuity and shock discontinuities, (2) positivity preserving of scalar quantities, (3) free of “carbuncle phenomenon”.

In the year 2006, Liou introduced a sequel scheme to the AUSM⁺ called the AUSM⁺-up [10] extended for all speed flows. The AUSM⁺-up is implemented in the **Arion** code and it is given as follows:

$$\mathbf{F}_c(\mathbf{Q}_l, \mathbf{Q}_r) = \mathbf{p}_{1/2} + \dot{m}_{1/2} \begin{cases} \boldsymbol{\psi}_l & \text{if } M_{1/2} > 0 \\ \boldsymbol{\psi}_r & \text{otherwise} \end{cases} \quad (4.14)$$

where

$$\boldsymbol{\psi}_{l/r} = \begin{cases} 1 \\ u_{l/r} \\ v_{l/r} \\ w_{l/r} \\ H_{l/r} \end{cases}, \quad (4.15)$$

the mass flux, $\dot{m}_{1/2}$ is defined as

$$\dot{m}_{1/2} = a_{1/2} M_{1/2} \begin{cases} \rho_l & \text{if } M_{1/2} > 0 \\ \rho_r & \text{otherwise} \end{cases}, \quad (4.16)$$



and the pressure flux, $\mathbf{p}_{1/2}$ is given as

$$\mathbf{p}_{1/2} = \begin{Bmatrix} 0 \\ p_{1/2}n_x \\ p_{1/2}n_y \\ p_{1/2}n_z \\ 0 \end{Bmatrix} \quad (4.17)$$

where

$$p_{1/2} = \mathcal{P}_{(5)}^+(M_l)p_l + \mathcal{P}_{(5)}^-(M_r)p_r - K_u \mathcal{P}_{(5)}^+(M_l) \mathcal{P}_{(5)}^-(M_r) (\rho_l + \rho_r) (f_a a_{1/2}) (q_r - q_l) \quad (4.18)$$

is the interface pressure. The interface-normal velocity is denoted by q , H denotes the specific total enthalpy, and K_u is a constant that equals 0.75. The remaining functions are given below. The left/right Mach number at the interface, $M_{l/r}$, is defined as follows:

$$M_{l/r} = \frac{q_{l/r}}{a_{1/2}} \quad (4.19)$$

where $a_{1/2}$ is the speed of sound at the interface and it may be calculated by a simple average of a_l and a_r :

$$a_{1/2} = \frac{a_l + a_r}{2} \quad (4.20)$$

Next, the Mach number at the interface, $M_{1/2}$ is calculated as follows:

$$\begin{aligned} \overline{M}^2 &= \frac{q_l^2 + q_r^2}{2a_{1/2}^2} \\ M_o^2 &= \min\left(1, \max\left(\overline{M}^2, M_\infty^2\right)\right) \\ f_a(M_o) &= M_o(2 - M_o) \\ \rho_{1/2} &= \frac{\rho_l + \rho_r}{2} \\ M_{1/2} &= \mathcal{M}_{(4)}^+(M_l) + \mathcal{M}_{(4)}^-(M_r) - \frac{K_p}{f_a} \max\left(1 - \sigma \overline{M}^2, 0\right) \frac{p_r - p_l}{\rho_{1/2} a_{1/2}^2} \end{aligned} \quad (4.21)$$

with the constants $K_p = 0.25$ and $\sigma = 1.0$. The split Mach numbers $\mathcal{M}_m^{+/-}$ are



polynomial functions of degree m ($=1,2,4$) of the Mach number, M , given as follows:

$$\begin{aligned}\mathcal{M}_1^\pm &= \frac{1}{2}(M \pm |M|) \\ \mathcal{M}_2^\pm &= \pm \frac{1}{4}(M \pm 1)^2 \\ \mathcal{M}_{(4)}^\pm(M) &= \left\{ \begin{array}{ll} \mathcal{M}_{(1)}^\pm & \text{if } |M| > 0 \\ \mathcal{M}_{(2)}^\pm(1 \mp 16\beta\mathcal{M}_{(2)}^\mp) & \text{otherwise} \end{array} \right\}\end{aligned}\quad (4.22)$$

with the constant $\beta = 1/8$. Finally, the pressure polynomials are given as:

$$\mathcal{P}_{(5)}^\pm(M) = \left\{ \begin{array}{ll} \frac{1}{M}\mathcal{M}_{(1)}^\pm & \text{if } |M| \geq 1 \\ \mathcal{M}_{(2)}^\pm[(\pm 2 - M) \mp 16\alpha M\mathcal{M}_{(2)}^\mp] & \text{otherwise} \end{array} \right\}\quad (4.23)$$

with the function $\alpha = \frac{3}{16}(-4 + 5f_a^2)$.

4.2.3 Passive Scalar Approach

The **Arion** code implements the passive scalar approach [4] in spatial discretization of the various model equations (*e.g.*, turbulence, finite-rate chemistry, etc.). The passive scalar approach enables to treat the extended governing equation set (including the model equations) in a similar manner to that presented for the Navier-Stokes equations. For example, when using any of the $k - \omega$ models with the HLLC scheme, the left and right state vectors are extended as follows:

$$\mathbf{Q}_l^* = \begin{pmatrix} \rho_l^* \\ (\rho u)_l^* \\ (\rho v)_l^* \\ (\rho w)_l^* \\ E_l^* \\ (\rho k)^* \\ (\rho \phi)^* \end{pmatrix} = \Omega_l \begin{pmatrix} \rho_l (S_l - q_l) \\ (S_l - q_l) (\rho u)_l + (p^* - p_l) n_x \\ (S_l - q_l) (\rho v)_l + (p^* - p_l) n_y \\ (S_l - q_l) (\rho w)_l + (p^* - p_l) n_z \\ (S_l - q_l) E_l - p_l q_l + p^* S_m \\ \rho k \\ \rho \phi \end{pmatrix}\quad (4.24)$$

The HLLC inviscid flux is then easily evaluated using:

$$\mathbf{H}_{c_l}^* \equiv \mathbf{H}_c(\mathbf{Q}_l^*) = \mathbf{Q}_l^* S_m + \begin{pmatrix} 0 \\ p^* n_x \\ p^* n_y \\ p^* n_z \\ p^* S_m \\ 0 \\ 0 \end{pmatrix} \quad (4.25)$$

4.2.4 High Order Flux Approximations

For a first-order-accurate approximation, the left and right state vectors are simply calculated from cell-center values left and right of the interface, respectively. To obtain a higher order flux approximation, the left and right state vectors of the convective flux are evaluated using a linear reconstruction using Green's theorem or a Taylor series expansion and least squares method. A cell-wise gradient of the primitive variables is constructed, followed by a second order Taylor series expansion, the left and right states are reconstructed. Let the vector $\mathbf{W} = (W_m; m = 1, \dots, 7)$ denote the primitive variables vector (for a general $k - \omega$ turbulence model),

$$\mathbf{W} = [\bar{\rho}, \tilde{u}, \tilde{v}, \tilde{w}, \bar{p}, k, \omega] \quad (4.26)$$

then the left and right primitive variables are reconstructed as follows:

$$(W_m)_L = (W_m)_i + (\psi_m)_i (\nabla W_m)_i \cdot \mathbf{d}_i^{ij} \quad (4.27a)$$

$$(W_m)_R = (W_m)_j + (\psi_m)_j (\nabla W_m)_j \cdot \mathbf{d}_j^{ij} \quad (4.27b)$$

where \mathbf{d}_i^{ij} (\mathbf{d}_j^{ij}) is the distance vector from the mid-point of face ij to the center of cell i (j), and ψ_m is the cell limiter that is used to suppress oscillations in the solution.



4.3 Diffusive Flux Vector Discretization

The diffusive flux vector normal to the interface, \mathcal{H}_d , is a function of the primitive variables vector, \mathcal{W} , evaluated at the mid-point of face ij , \mathcal{W}_{ij} , and of its derivatives. The vector \mathcal{W}_{ij} is calculated by averaging of adjacent cell-center values (*i.e.*, \mathcal{W}_i and \mathcal{W}_j).

4.4 Time Marching Schemes

The **Arion** code provides various possibilities for advancing Equation (4.3) in time. This section contains a brief description of the available schemes. The schemes may be classified as follows:

1. Explicit (single and multi stage) schemes
 - (a) Explicit Euler
 - (b) Third and fourth order Runge-Kutta schemes
2. Implicit schemes
 - (a) Point Gauss-Seidel
3. Multi-stage implicit schemes
 - (a) Third fourth and fifth order Runge-Kutta implicit schemes

4.4.1 Explicit schemes

4.4.1.1 Explicit Euler Scheme

Consider the semi-discrete equation¹:

$$V \frac{dQ}{dt} = R \tag{4.28}$$

¹Equation (4.3) without the index

A simple, first-order Euler explicit time marching scheme is given by:

$$\Delta Q^n = \frac{\Delta t}{V} R^n \quad (4.29)$$

where ΔQ^n is the increment of the solution between time levels , namely,

$$\Delta Q^n = Q^{n+1} - Q^n \quad (4.30)$$

4.4.1.2 Runge-Kutta Schemes

Arion provides the choice of third or fourth order Runge-Kutta schemes. Consider the semi-discrete formulation as presented in Equation (4.28), the Runge-Kutta scheme formulation is as follows:

$$\begin{aligned} Q^{(0)} &= Q^n \\ Q^{(k)} &= Q^n + \alpha_k \frac{\Delta t}{V} R^{(k-1)}, \quad k = 1, \dots, K \\ Q^{n+1} &= Q^{(K)} \end{aligned} \quad (4.31)$$

where k is the Runge-Kutta sub-step number, $K = 3$ for third order and $K = 4$ for fourth order, and α_k are the appropriate weights.

4.4.2 Implicit Time Marching Formulation

The fine grid spacing required to resolve the normal viscous terms close to the body surface requires in turn very small time steps and therefore it rules out the use of explicit methods. In explicit time-marching schemes the maximum time step is proportional to the minimum grid spacing. As a result the time-step limit imposed by stability is very small. In contrast, even though the operation count per time step is high, it is more efficient to use implicit methods. The development of a non-iterative implicit algorithm for the solution of the Navier-Stokes equations requires a time linearization of the nonlinear vectors (R). The linearization procedure is simple since the equations are written in conservation-law form. Applying the first order Euler implicit method and utilizing the Delta form of the equations results in the following

implicit scheme:

$$\left(\frac{V}{\Delta t}I - \frac{\partial R}{\partial Q}\right)^n \Delta Q^n = R^n \quad (4.32)$$

where R^n is the residual at time level n as defined by Equation (4.3), I is the identity matrix, Δt is the time increment between levels n and $n + 1$, the term $\frac{\partial R}{\partial Q}$ is the Jacobian matrix. Note that the Jacobian matrix is first order and that it may be altered to improve stability.

Applying Equation (4.32) at every grid point results in a block-hepta-diagonal matrix in three dimensions. The inversion of the matrix, or its approximation, may be conducted in various manners, resulting in a wide variety of implicit time marching schemes.

4.4.2.1 Point Gauss-Seidel

The exact, first order Jacobian matrix is retained only for the diagonal elements and the off diagonal Jacobian matrices are linearized based on the previous time step as follows:

$$\frac{\partial R}{\partial Q} \Delta Q \approx \left(\frac{\partial R}{\partial Q}\right)^n \Delta Q^n \quad (4.33)$$

Consequently, they can be moved to the right hand side.

4.4.2.2 Runge-Kutta Schemes

Runge-Kutta implicit schemes combine the explicit Runge-Kutta schemes as described in Section 4.4.1.2 with the PGS scheme that is described in Section 4.4.2.1 to form a robust implicit, multi-stage time marching scheme. Note that this scheme requires the matrix inversions warranted by the PGS scheme at each stage and therefore requires more computer time per iteration.

Chapter 5

Boundary Conditions

5.1 Introduction

The **Arion** code contains a wide variety of boundary conditions. Being an unstructured finite volume code, the notion of a ghost cell is utilized throughout. However, in certain cases the Jacobian and flux are explicitly dictated rather than calculated based on the ghost. In particular, the convection Jacobian and flux. In what follows, the subscript “*g*” signifies a ghost cell, the subscript “*r*” signifies a real cell where the flow is solved (a “real” cell), and the subscript “*f*” signifies the face (prescribed) value.

5.2 Wall Boundary Conditions

5.2.1 Impermeable Wall Conditions

Let \hat{n} be a unit vector normal to the face of a boundary cell, whose components are (n_x, n_y, n_z) , and let t^1 and t^2 be the two unit vectors tangent to the face of a boundary



cell, the velocity components in the ghost cell are calculated by solving the system:

$$\begin{bmatrix} n_x & n_y & n_z \\ t_x^1 & t_y^1 & t_z^1 \\ t_x^2 & t_y^2 & t_z^2 \end{bmatrix} \begin{pmatrix} u_g \\ v_g \\ w_g \end{pmatrix} = \begin{pmatrix} 2\bar{V}_f \cdot \hat{n} - \bar{V}_r \cdot \hat{n} \\ \bar{V}_r \cdot \hat{t}^1 \\ \bar{V}_r \cdot \hat{t}^2 \end{pmatrix} \quad (5.1)$$

The prescribed velocity vector \bar{V}_f includes any motion of the boundary surface.

5.2.2 No-Slip Condition

The no-slip condition is much easier to implement:

$$\bar{V}_g = 2\bar{V}_f - \bar{V}_r \quad (5.2)$$

5.2.3 Adiabatic Wall

The temperature is set using:

$$T_g = T_r \quad (5.3)$$

while the pressure is set using

$$p_g = p_r \quad (5.4)$$

The density is evaluated using the equation of state.

5.3 Far Field Conditions

5.3.1 Turkel Type Conditions

The characteristic relations that are due-to Turkel are utilized. For supersonic inflow, the flow quantities at the inflow boundary are set based on the current values of the corresponding boundary:



5.3.1.1 Turkel Inlet

$$\rho_g = \rho_\infty \quad (5.5)$$

$$\bar{V}_g = \bar{V}_\infty \quad (5.6)$$

$$p_g = \frac{\rho_\infty}{\rho_r} p_r \quad (5.7)$$

5.3.1.2 Turkel Outlet

$$\rho_g = \rho_r + \frac{p_\infty - p}{a_\infty^2} \quad (5.8)$$

$$\bar{V}_g = \bar{V}_r \quad (5.9)$$

$$p_g = p_\infty \quad (5.10)$$

5.3.2 Riemann Type Conditions

Let q be the normal to the boundary face velocity. The Riemann invariants are calculated based on the following:

$$\begin{aligned} R^+ &= q_r - \frac{2}{\gamma - 1} a_\infty \\ R^- &= q_\infty - \frac{2}{\gamma - 1} a_r \end{aligned} \quad (5.11)$$

The ghost Riemann invariants are obtained using:

$$R_g = \frac{1}{2} (R^+ + R^-) \quad (5.12)$$

5.3.2.1 Riemann Inlet

$$\begin{aligned} \rho_g &= \frac{\rho_\infty}{a_\infty^{\frac{1}{\gamma-1}}} \left\{ \underbrace{\left[\frac{\gamma-1}{4} (R^- - R^+) \right]^{\frac{1}{\gamma-1}}}_{a_g} \right\}^2 \\ p_g &= \sqrt{a_g} \rho_g \gamma \end{aligned} \quad (5.13)$$

5.3.2.2 Riemann Outlet

Let s be the entropy, the relations for ρ_g and p_g are given by:

$$\begin{aligned}s &= \frac{\rho^\gamma}{\gamma p} \\ \rho_g &= (a_g^2 s)^{\frac{1}{\gamma-1}} \\ p_g &= \sqrt{a_g} \rho_g \gamma\end{aligned}\tag{5.14}$$

5.3.3 Fixed (Supersonic Inlet)

$$\begin{aligned}\bar{V}_g &= \bar{V}_\infty \\ p_g &= p_\infty \\ T_g &= T_\infty\end{aligned}\tag{5.15}$$

5.3.4 Extrapolation (Supersonic Outlet)

$$\begin{aligned}\bar{V}_g &= \bar{V}_r \\ p_g &= p_r \\ T_g &= T_r\end{aligned}\tag{5.16}$$

5.3.5 Inlet

These boundary conditions use extrapolation for the pressure and set to “Fixed” the rest of the variables.

$$\begin{aligned}(\rho, u, v, w)_g &= (\rho, u, v, w)_\infty \\ p_g &= p_r\end{aligned}\tag{5.17}$$

5.3.6 Outlet

These boundary conditions use a “Fixed” value for the pressure and use extrapolation for the rest of the variables.

$$\begin{aligned}(\rho, u, v, w)_g &= (\rho, u, v, w)_r \\ p_g &= p_\infty\end{aligned}\tag{5.18}$$

5.3.7 Inout

These boundary conditions are specific to low subsonic flows. With the exception of the pressure, these boundary conditions set “Fixed” conditions for inlet and “Extrapolation” conditions for outlet. Namely,

- **Inlet :**

$$\begin{aligned}(\rho, u, v, w)_g &= (\rho, u, v, w)_\infty \\ p_g &= p_r\end{aligned}\tag{5.19}$$

- **Outlet :**

$$\begin{aligned}(\rho, u, v, w)_g &= (\rho, u, v, w)_r \\ p_g &= p_\infty\end{aligned}\tag{5.20}$$

5.4 Symmetry Boundary Conditions

Symmetric boundary conditions are treated exactly as an adiabatic impermeable wall.



Chapter 6

Computational Mesh

The **Arion** code is considered a hybrid code since it support various cell element types. The code supports tetrahedra, hexahedra, prism, and pyramids. Mesh generation may be conducted by any unstructured grid generator, however, users must export the mesh using a Star-CD export. The export results in three files. The first, a file containing the vertex information, having the extension “.vrt.” The second, a file containing the cell elements, having the extension “.cel.” And finally, a file containing the boundary elements (triangles or quads), having the extension “.bnd.” The “.bnd” file contains the boundary elements as well as a name for each element. The naming is conducted by the user using the grid generation package (*e.g.*, Pointwise). The code supports two variations of the Star-CD format, the export by Pointwise and the export by CENTAUR (only 3-D export is supported).



Chapter 7

Parallelization

The **Arion** flow solver is designed to work in a distributed memory architecture using the MPI interface. The design of the code distinguishes between managing a simulation and solving the flow field. Within the distributed MPI universe, the first rank is responsible for managing the simulation and henceforth named ‘manager rank’. The rest of the ranks are responsible for the flow solution and are named ‘worker ranks’. The manager rank responsibility starts with the input analysis, and continuing with reading the initial geometric problem (the grid files), splitting the computational domain into parts and sending those parts to the worker ranks. Therefore, the worker ranks know only part of the computational domain, and pass boundary data among themselves. The manager rank is responsible for the the assembly of the ‘restart’ files, for timing all the worker threads, and for log output. Since there is a distinction between the manager rank and the worker ranks, one must have at least two ranks running, even if they reside on a single shared memory machine.



Chapter 8

Input File, Run Preparation, and Execution

8.1 Preface

The **Arion** code is driven through the command line with optionally additional input files having a certain syntax. An input file may be constructed using a simple text editor. The solver is invoked by typing the MPI command (depending on the MPI version of the actual machine): “`mpirun [mpi options]... arion [--f input_file] [command-line arguments]`”

The input file is made of groups of directives, each group is marked by two consecutive `-` signs. Each group has a series of options, with each option marked with one `-` sign. Within the input file the sign `!` means a remark until the end of the line. A brief help of all the options may be printed to the screen by using the `--h` option. The latest additions to the code may be printed to the screen using the `--new` option.



8.1.1 Scripting Language

The input file is a simple ASCII file with certain important rules. This chapter contains a detailed description of all the available command-line options. The syntax of each of the input options, or input file lines, follows the proceeding rules:

- The “#” symbol means that there is a comment until the next option. One may use as many comment lines as necessary.
- Each group of directives starts with two consecutive – signs, *i.e.*, -- followed by the group name and a series of options.
- Each option is marked with one – sign.
- An option may have no parameters, one parameter, or a few parameters.
- Parameters may be assigned values.

The proceeding description of the input options, or input file lines, makes use of the following symbols:

- The “\$” symbol signifies a string input.
- The “#” symbol signifies a numerical input.
- The “|” symbol signifies a choice selection.
- The “...” symbol signifies an option that can be repeated.
- Input entries that are enclosed by curly brackets, { }, are required.
- Input entries that are enclosed by square brackets, [], are optional.

8.1.2 Basic Input File: Example

Listing 8.1 contains a simple, basic input file for a simple simulation of the ideal-gas flow about a two-dimensional airfoil.

```
--equation.of.state
  -name air
  -type eos.ideal.gas
  -ref.p 101325
  -ref.T 288

--bc
  -name RIE_FREE
  -type riemann

--bc
  -name IWALL
  -type impermeable.wall

--bc
  -name XZSYM
  -type 2d

--system
  -name ideal.gas.mf.inviscid
  -type ideal.gas.mf.inviscid
  -time.step 5 100 cfl.exponential 250
    -cell.gradient green.gauss.node
    -limiter mlp.2d

  -log convergence cfl residual log.residual
```

```
--solve
-time.march      implicit.rk3.R.pgs
-convection.flux  hllc.roe
-convection.jacobian hllc.roe
-sweeps 4
-spatial.order 2

-iterations 2000
-save 100
-save.path ./save/

-eos air
-p 101325
-T 288
-Mach 0.8
-alpha 1.25
-beta 0
-log convergence iter

--log
-name convergence
-prefix ./logs/convergence
-per iter
-screen

--plot
-name naca0012
-prefix ./plots/plot
-interval 100
```

```
--uns
  -name naca0012-j129-uns-regular
  -prefix naca0012-j129-uns-regular/naca0012-j129-uns-regular
  -log convergence cl cd

--table
  -name P_table
  -prefix ./tables/table
  -interval 10
  -plane.bc 0 0 0 0 0.5 0 IWALL
```

Listing 8.1: Example of a simple **Arion** input file

8.2 Star-CD Grid Files

Currently, **Arion** supports the Star-CD export only. A detailed description of the files is given in Chapter 6. Starting from Version 1.31, **Arion** uses the Metis open source library to convert the Star-CD export to binary format. In addition, Metis assists in decomposing the computational domain into several partitions for the purpose of efficient parallel computations. The following section describes the use of the Metis library for conversion of the files to binary format. The conversion results in a new set of files with “+32” added to all file extensions, signifying that the integers are 32 bit wide.

The new set of files now contains 5 files. The vertex information has the extension “.vrt+32.” The cell elements file has the extension “.cel+32.” The boundary elements file has the extension “.bnd+32.” In addition, two new files are generated, one with the extension “.nam+32” and the other with the extension “.fac+32.” As a result, the grid is read by the code in a fast manner.

8.3 Conversion

The conversion is conducted using the `--star2metis` directive as described in Table 8.1. Note that the Metis options that are described in Tables 8.1 and ... are taken directly from the Metis runtime help. For further information the user is referred to the complete Metis User's Manual.

The conversion generates a new set of files, replacing the Star-CD files that were generated by the grid generation software package (*e.g.*, Pointwise). The actual files are not split but are reordered in a manner that prepares the files for a parallel read that is efficient in terms of domain decomposition.

Conversion							
Syntax	<code>arion --star2metis \$prefix #ranks \$output [metis-options]...</code>						
Description	Convert the Star-CD files binary format and use Metis to decompose the computational domain. See Table 8.2 for Metis options.						
Parameters	<table> <tr> <td><code>\$prefix</code></td> <td>Prefix of Star-CD files.</td> </tr> <tr> <td><code>#ranks</code></td> <td>Number of partitions.</td> </tr> <tr> <td><code>\$output</code></td> <td>Prefix of output files.</td> </tr> </table>	<code>\$prefix</code>	Prefix of Star-CD files.	<code>#ranks</code>	Number of partitions.	<code>\$output</code>	Prefix of output files.
<code>\$prefix</code>	Prefix of Star-CD files.						
<code>#ranks</code>	Number of partitions.						
<code>\$output</code>	Prefix of output files.						
Examples	<code>arion --star2metis Star-CD-file-name 64 New-file-name</code>						

Table 8.1: Conversion



Metis Options	
Metis Option	Description
-ptype {rb kway}	Specify the scheme to be used for computing the k-way partitioning: rb - Recursive bisectioning, kway - Direct k-way partitioning [default].
-ctype {rm shem}	Specify the scheme to be used to match the vertices of the graph; rm - Random matching, shem - Sorted heavy-edge matching [default].
-iptype {grow random}	Specify the scheme to be used to compute the initial partitioning, (applies only when -ptype = rb); grow - Grow a bisection using a greedy scheme [default], random - Compute a bisection at random.
-objtype {cut vol}	Specify the objective that the partitioning routines will optimize, (applies only when -ptype = kway); cut - Minimize the edgecut [default]; vol - Minimize the total communication volume.
-no2hop	Specify that the coarsening will not perform any 2-hop matchings when the standard matching fails to sufficiently contract the graph.
-contig	Specify that the partitioning routines should try to produce (applies only when -ptype = kway) partitions that are contiguous. Note that if the input graph is not connected this option is ignored.
-minconn	Specify that the partitioning routines should try to minimize the (applies only when -ptype = kway) maximum degree of the subdomain graph, i.e., the graph in which each partition is a node, and edges connect subdomains with a shared interface.

Table 8.2: Metis options

Metis Options (continued)

Metis Option	Description
-ufactor #x (where x is an integer)	Specify the maximum allowed load imbalance among the partitions. A value of x indicates that the allowed load imbalance is $1+x/1000$. For -ptype=rb, the load imbalance is measured as the ratio of the $2*\max(\text{left},\text{right})/(\text{left}+\text{right})$, where left and right are the sizes of the respective partitions at each bisection. For -ptype=kway, the load imbalance is measured as the ratio of $\max_i(\text{pwgts}[i])/\text{avgpwgt}$, where $\text{pwgts}[i]$ is the weight of the i th partition and avgpwgt is the sum of the total vertex weights divided by the number of partitions requested. For -ptype=rb, the default value is 1 (i.e., load imbalance of 1.001) For -ptype=kway, the default value is 30 (i.e., load imbalance of 1.03).
-niter #i	Specify the number of iterations for the refinement algorithms at each stage of the uncoarsening process. Default is 10.
-ncuts #c	Specify the number of different partitionings that it will compute. The final partitioning is the one that achieves the best edgecut or communication volume. Default is 1.
-seed #s	Select the seed of the random number generator.

Table 8.2: Metis options (continued)

8.3.1 Conversion Between Different Orderings

Arion provides the capabilities to start a simulation from a certain ordering and continue the simulation with a different ordering. This is conducted using the `--cnv2cnv` directive as described in Table 8.3.

cnv2cnv											
Syntax	<code>arion --cnv2cnv \$cnv1 \$cnv2 \$load-prefix \$save-prefix {\$system1-name}...</code>										
Description	Converts Arion solution files between different orderings; creates new restart files.										
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="padding-right: 20px;"><code>\$cnv1</code></td> <td>Ordering type to convert from.</td> </tr> <tr> <td><code>\$cnv2</code></td> <td>Ordering type to convert to.</td> </tr> <tr> <td><code>\$load-prefix</code></td> <td>Prefix of input files.</td> </tr> <tr> <td><code>\$save-prefix</code></td> <td>Prefix of output files.</td> </tr> <tr> <td><code>\$system1-name</code></td> <td>Name of converted system.</td> </tr> </table>	<code>\$cnv1</code>	Ordering type to convert from.	<code>\$cnv2</code>	Ordering type to convert to.	<code>\$load-prefix</code>	Prefix of input files.	<code>\$save-prefix</code>	Prefix of output files.	<code>\$system1-name</code>	Name of converted system.
<code>\$cnv1</code>	Ordering type to convert from.										
<code>\$cnv2</code>	Ordering type to convert to.										
<code>\$load-prefix</code>	Prefix of input files.										
<code>\$save-prefix</code>	Prefix of output files.										
<code>\$system1-name</code>	Name of converted system.										
Examples	<code>arion --cnv2cnv bin arion ./newsolution/oldfiles ./newsolution/newfiles ./restart/oldfiles</code>										

Table 8.3: Conversion between different ordering (`cnv2cnv`) option



8.4 Log Control Options

Log control options are used to set up the log outputs. For example, the options are used to set up the log file name. The log control options section starts with the `--log` directive, followed by a series of options that are described in the following tables. It is important to note that the number of logs (and thus generated log files) is unlimited.

Name	
Syntax	<code>--name \$name</code>
Description	Sets the name of the log. This name is referred to by the specific <code>--log</code> options. It can be the predefined keywords: 'stdout', 'screen', or 'display': where it will be only shown on the screen. It can also be: 'stderr' where it will be only directed to the standard error stream.
Parameters	<code>\$name</code> Log name.
Examples	<code>--name convergence</code>

Table 8.4: Log name option



Prefix	
Syntax	<code>–prefix \$prefix</code>
Description	The log file path and name prefix. The 'log' suffix is added to obtain the actual log file name.
Parameters	<code>\$prefix</code> Prefix.
Examples	<code>–prefix ./logs/convergence</code>

Table 8.5: Log prefix option

Per	
Syntax	<code>–per \$type</code>
Description	Sets when the log is printed based on the \$type. The possibilities are: iter, produced every iteration (virtual time step); step, produced every step (physical time step); or pos, produced every 'pos' (depending on the simulation type).
Parameters	<code>\$type</code> Type (iter, step, or pos).
Examples	<code>–per pos</code>

Table 8.6: Log per option



Stdout	
Syntax	–stdout
Description	Adds echo to the standard output as well as to the log file. One can also use the -screen or -display (the outcome is exactly the same).
Parameters	N/A
Examples	–screen

Table 8.7: Log stdout option

Stderr	
Syntax	–stderr
Description	Adds echo to the standard error as well as to the log file.
Parameters	N/A
Examples	–stderr

Table 8.8: Log stderr option



8.5 Plot Control Options

Plot control options are used to set up the plot outputs. The plot control options section starts with the `--plot` directive, followed by a series of options that are described in the following tables.

Name	
Syntax	<code>--name \$name</code>
Description	Sets the name of the plot. This name is referred to by the specific <code>--plot</code> options.
Parameters	<code>\$name</code> Plot name.
Examples	<code>--name Alpha10</code>

Table 8.9: Plot name option

Prefix	
Syntax	<code>--prefix \$prefix</code>
Description	The plot file path and name prefix. The actual file number is composed from the prefix, and the <code>\$name</code> as described in Table 8.118. The <code>'fvuns'</code> suffix is added to the plot file name. If the computational domain has been decomposed, the files are split as well and the number of the partition is added to the file name.
Parameters	<code>\$prefix</code> Prefix.
Examples	<code>--prefix ./plots/</code>

Table 8.10: Plot prefix option



Interval	
Syntax	<code>–interval #interval</code>
Description	Sets the interval (of iterations/steps) to create the plot files.
Parameters	<code>#interval</code> Plot interval.
Examples	<code>–interval 100</code>

Table 8.11: Plot interval option

Sequential/overwrite	
Syntax	<code>–sequential / –overwrite</code>
Description	Include position stamp in the plot files (or overwrite, without position stamp, default).
Parameters	N/A
Examples	<code>–sequential</code>

Table 8.12: Plot sequential/overwrite option



FVUNS/EPPIC	
Syntax	<code>-fvuns / -eppic</code>
Description	Select the file format to use, default is:-fvuns.
Parameters	N/A
Examples	<code>-eppic</code>

Table 8.13: Plot fvuns/eppic option



8.6 Table Control Options

Table control options are used to set up the table outputs. The table control options section starts with the `--table` directive, followed by a series of options that are described in the following tables.

Name	
Syntax	<code>--name \$name</code>
Description	Sets the name of the table. This name is referred to by the specific <code>--table</code> options.
Parameters	<code>\$name</code> Table name.
Examples	<code>--name Alpha10</code>

Table 8.14: Table name option

Prefix	
Syntax	<code>--prefix \$prefix</code>
Description	The table file path and name prefix. The '.tbl' suffix is added to obtain the actual table file name.
Parameters	<code>\$prefix</code> Prefix.
Examples	<code>--prefix ./tables/surface-pressure</code>

Table 8.15: Table prefix option



Interval	
Syntax	<code>-interval #interval</code>
Description	Sets the interval (of iterations/steps) to create the table files.
Parameters	<code>#interval</code> Table interval.
Examples	<code>-interval 100</code>

Table 8.16: Table interval option

Sequential/overwrite	
Syntax	<code>-sequential / -overwrite</code>
Description	Include position stamp in the table files (or overwrite, without position stamp, default).
Parameters	N/A
Examples	<code>-sequential</code>

Table 8.17: Table sequential/overwrite option

Horizontal/vertical	
Syntax	<code>-horizontal / -vertical</code>
Description	The table type (vertical is the default).
Parameters	N/A
Examples	<code>-horizontal</code>

Table 8.18: Table horizontal/vertical option



Ray					
Syntax	<code>-ray #x #y #z #vx #vy #vz</code>				
Description	Create a table along a predefined ray.				
Parameters	<table border="0"> <tr> <td><code>#x #y #z</code></td> <td>Ray origin.</td> </tr> <tr> <td><code>#vx #vy #vz</code></td> <td>Ray direction.</td> </tr> </table>	<code>#x #y #z</code>	Ray origin.	<code>#vx #vy #vz</code>	Ray direction.
<code>#x #y #z</code>	Ray origin.				
<code>#vx #vy #vz</code>	Ray direction.				
Examples	<code>-ray 0 0 0 1 1 1</code>				

Table 8.19: Table ray option

Plane.bc							
Syntax	<code>-plane.bc #x #y #z #nx #ny #nz \$bc-name</code>						
Description	Create a table along an intersection between a plane and a boundary surface.						
Parameters	<table border="0"> <tr> <td><code>#x #y #z</code></td> <td>Plane.bc origin.</td> </tr> <tr> <td><code>#nx #ny #nz</code></td> <td>Plane.bc direction.</td> </tr> <tr> <td><code>\$bc-name</code></td> <td>The boundary surface name as assigned by the <code>--bc -name</code> directive (see Table 8.38).</td> </tr> </table>	<code>#x #y #z</code>	Plane.bc origin.	<code>#nx #ny #nz</code>	Plane.bc direction.	<code>\$bc-name</code>	The boundary surface name as assigned by the <code>--bc -name</code> directive (see Table 8.38).
<code>#x #y #z</code>	Plane.bc origin.						
<code>#nx #ny #nz</code>	Plane.bc direction.						
<code>\$bc-name</code>	The boundary surface name as assigned by the <code>--bc -name</code> directive (see Table 8.38).						
Examples	<code>-plane.bc 0 0 0 1 1 1 WALL</code>						

Table 8.20: Table plane.bc option



8.7 Parallel Options

The parallel input options section starts with the `--parallel` directive, followed by a series of parallel options.

Cache	
Syntax	<code>--cache #</code>
Description	Size of the cache memory (in bytes).
Parameters	<code>#</code> Cache size.
Examples	<code>--cache 1000</code>

Table 8.21: Parallel cache option



Rank		
Syntax	[{-rank #rank -all.ranks} {-threads -parts -load -read.threads} #]...	
Description	Define how many OpenMP threads will be open on each rank.	
Parameters	#rank	Rank number.
	-threads	Number of threads for the current rank.
	-all.ranks	Means that all subsequent options hold for all the ranks.
	-threads	Sets the number of solver threads on a specific (or all) rank.
	-read.threads	Use when memory is insufficient to read all the part in parallel (a significant amount of memory is needed to read a zone part, that is deallocated at the end of the read).
	-parts	Number of partitions.
	-load	Changes the load of a rank, by default all ranks have load 1.
	#	Load level for the current rank.
Examples	-rank 1 -threads 1 -rank 2 -threads 8 -all.ranks -threads 48 -rank 2 -load 4	

Table 8.22: Parallel rank option



8.8 Equation of State Options

The “Equation of State” input options sections start with the `--equation.of.state` directive, followed by a series of `equation.of.state` input options. Each section is used to declare a specific fluid with its equation of state and constitutive laws (*e.g.* Sutherland’s law). There could be more than one section, one per fluid type. It is first assigned a name to be used by the user. The current version of **Arion** provides the means to solve only a single fluid and therefore only a single equation of state is concurrently supported. If nothing is set then the default ideal gas parameters of air are used.

Name	
Syntax	<code>--name \$fluid</code>
Description	Name of flow medium.
Parameters	<code>\$fluid</code> Flow medium.
Examples	<code>--name air</code>

Table 8.23: `equation.of.state` name option



Type	
Syntax	<code>-type \$eos.type.fluid</code>
Description	Choose the type of equation of state. Available type: <code>eos.ideal.gas</code> (see Section 2.2.3 and Equation 2.10)
Parameters	<code>\$eos.type.fluid</code> Type of equation of state.
Examples	<code>-type eos.ideal.gas</code>

Table 8.24: equation.of.state type option

R	
Syntax	<code>-R #</code>
Description	Set the specific gas constant. Default is 287.22 (air).
Parameters	<code>#</code> Specific gas constant.
Examples	<code>-R 287.22</code>

Table 8.25: equation.of.state R option

Gamma	
Syntax	<code>-gamma #</code>
Description	Set the heat capacity ratio (γ). Default is 1.4 (air).
Parameters	<code>#</code> Heat capacity ratio.
Examples	<code>-gamma 1.4</code>

Table 8.26: equation.of.state gamma option



Ref		
Syntax	-ref.\$ #	
Description	Sets reference properties. These reference values are relevant in particular for the limiters.	
Parameters	-ref.p	Reference pressure. Default is 1.
	-ref.T	Reference temperature. Default is 1.
	-ref.length	Reference length. Default is 1.
	#	Reference value.
Examples	-ref.p 101325	
	-ref.length 0.5	

Table 8.27: equation.of.state ref option



8.9 Molecular

The “Molecular” input options sections start with the `--molecular` directive, followed by a series of molecular input options. Each section is used to declare specific type or constants for Sutherland’s law. The Sutherland formulae can be set in two ways: The first way, called `molecular.sutherland.air.1`, is direct coefficients for μ and κ (see Equation 2.12)

Name	
Syntax	<code>--name \$name</code>
Description	Set the name of the molecular thermodynamic relations.
Parameters	<code>\$name</code> Name of molecular thermodynamic relations.
Examples	<code>--name therm1</code>

Table 8.28: Molecular name option

Type	
Syntax	<code>--type \$type</code>
Description	Set the type of the molecular thermodynamic relations. Available types are: <code>molecular.sutherland.air.1</code> and <code>molecular.sutherland.air.2</code> (see Section 2.2.4, Equation 2.12).
Parameters	<code>\$type</code> Type of the molecular thermodynamic relations.
Examples	<code>--type molecular.sutherland.air.1</code>

Table 8.29: molecular type option



Mu1	
Syntax	–Mu1 #Cmu1
Description	Sets the first coefficient for the Sutherland molecular viscosity formula; Default is 1.458e-06 (air, see Equation 2.12 and Table 2.1).
Parameters	#Cmu1 First coefficient for the Sutherland formula
Examples	–Mu1 1.458e-06

Table 8.30: Molecular Mu1 option

Mu2	
Syntax	–Mu2 #Cmu2
Description	Sets the second coefficient for the Sutherland molecular viscosity formula; Default is 110.3 (air, see Equation 2.12 and Table 2.1).
Parameters	#Cmu2 Second coefficient for the Sutherland formula
Examples	–Mu2 110.3

Table 8.31: Molecular Mu2 option



Ka1	
Syntax	–Ka1 #Cka1
Description	Sets the first coefficient for the Sutherland thermal conductivity formula; Default is 2.495e-03 (air, see Equation 2.12 and Table 2.1).
Parameters	#Cka1 First coefficient for the Sutherland formula
Examples	–Ka1 2.495e-03

Table 8.32: Molecular Ka1 option

Ka2	
Syntax	–Ka2 #Cka2
Description	Sets the second coefficient for the Sutherland heat conductivity formula; Default is 194.0 (air, see Equation 2.12 and Table 2.1).
Parameters	#Cka2 Second coefficient for the Sutherland formula
Examples	–Ka2 194.0

Table 8.33: Molecular Ka2 option

Eta0	
Syntax	–Eta0 #Eta0
Description	Sets η_0 for the Sutherland molecular viscosity formula; Default is 1.827×10^{-5} (air, see Equation 2.13 and Table 2.2).
Parameters	#Eta0 Coefficient for the Sutherland formula
Examples	–Eta0 1.827×10^{-5}

Table 8.34: Molecular Eta0 option



C0	
Syntax	<code>-C0 #C0</code>
Description	Sets C_0 for the Sutherland molecular viscosity formula; Default is 120 (air, see Equation 2.13 and Table 2.2).
Parameters	<code>#C0</code> Coefficient for the Sutherland formula
Examples	<code>-C0 120</code>

Table 8.35: Molecular C0 option

T0	
Syntax	<code>-T0 #T0</code>
Description	Sets T_0 for the Sutherland molecular viscosity formula; Default is 291.15 (air, see Equation 2.13 and Table 2.2).
Parameters	<code>#T0</code> Coefficient for the Sutherland formula
Examples	<code>-T0 291.15</code>

Table 8.36: Molecular T0 option

Prandtl	
Syntax	<code>-Prandtl #Prandtl</code>
Description	Sets the Prandtl number; Default is 0.72 (air, see Equation 2.13 and Table 2.2).
Parameters	<code>#Prandtl</code> Prandtl number
Examples	<code>-Prandtl 0.72</code>

Table 8.37: Molecular Prandtl option



8.10 BC Options

Boundary conditions are set for each boundary cell, based on the name that has been assigned by the user (see Chapter 6). For each name, representing a group of boundary cells, the boundary conditions are set in two steps. First, assign the name, and second, assign the type. Each group starts with the `--bc` directive, followed by the name and type as follows:

Name	
Syntax	<code>--name \$name</code>
Description	Name of the boundary as set in the Star-CD input files.
Parameters	<code>\$name</code> Name of boundary.
Examples	<code>--name WALL</code>

Table 8.38: BC name option

Type	
Syntax	<code>--type \$type</code>
Description	Type of boundary condition. See Chapter 5 and Table 8.40 for available types.
Parameters	<code>\$type</code> Type of boundary condition.
Examples	<code>--type noslip.wall</code>

Table 8.39: BC type option



Boundary Condition Types

BC type	Description (see Chapter 5 for details)
impermeable.wall	Impermeable wall.
noslip.wall	No slip wall.
automatic.wall.funct	Automatic wall function.
nichols.wall.function	Nichols type wall function.
riemann	Riemann type outer boundary conditions.
riemann.inlet	Riemann type inlet boundary conditions.
riemann.outlet	Riemann type outlet boundary conditions.
turkel	Turkel type outer boundary conditions.
turkel.inlet	Turkel type inlet boundary conditions.
turkel.outlet	Turkel type outlet boundary conditions.
inlet	Inlet type outer boundary conditions.
outlet	outlet type outer boundary conditions.
inout	Inout outer boundary conditions.
extrap	Zero order extrapolation.
fixed	Fixed boundary conditions.
symmetry	Symmetry type conditions.
2d	Two dimensional domain conditions.

Table 8.40: Boundary condition types



Log	
Syntax	<code>-log \$log [log-options] ...</code>
Description	Report in log file. See Table 8.42 for log options list.
Parameters	<code>\$log</code> Log name.
Examples	<code>-log coefficients cl cd</code>

Table 8.41: Boundary conditions log option



BC Log Options	
Log type	Description
wet.surface	Wet surface area.
fx	X coordinate direction force.
fy	Y coordinate direction force.
fz	Z coordinate direction force.
lift	Lift force.
drag	Drag force.
mx	Moment about X coordinate direction.
my	Moment about Y coordinate direction.
mz	Moment about Z coordinate direction.
cfx	X coordinate direction force coefficient.
cfy	Y coordinate direction force coefficient.
cfz	Z coordinate direction force coefficient.
cmx	Moment coefficient about X coordinate direction.
cmx	Moment coefficient about Y coordinate direction.
cmz	Moment coefficient about Z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.

Table 8.42: BC log options



8.10.1 Wall Distance Calculations

The following directives pertain to wall type boundary conditions. Normally, one would like to calculate wall distance based on whether a specific boundary has been assigned wall conditions. However, sometimes it is required to override this default. Note that the wall distance is used only for turbulence models that require wall distance. The following two directives are exactly for that. They should be used along with a wall boundary condition.

Wall.distance	
Syntax	<code>-wall.distance</code>
Description	Explicitly set wall distance search for a specific boundary. Default: for all walls.
Parameters	N/A
Examples	<code>-wall.distance</code>

Table 8.43: BC wall.distance option

No.wall.distance	
Syntax	<code>-no.wall.distance</code>
Description	Explicitly disable wall distance search for a specific boundary. Default: for all boundaries other than walls.
Parameters	N/A
Examples	<code>-no.wall.distance</code>

Table 8.44: BC no.wall.distance option



8.10.2 Freestream Conditions Override

Mach	
Syntax	–Mach #M
Description	Override the freestream Mach number.
Parameters	#M Freestream Mach number.
Examples	–M 0.95

Table 8.45: BC Mach option

Velocity.magnitude	
Syntax	–velocity.magnitude #V
Description	Override the freestream velocity [m/s].
Parameters	#V Freestream velocity [m/s].
Examples	–V 200

Table 8.46: BC velocity.magnitude option



Alpha	
Syntax	–alpha #aoa
Description	Override the freestream angle of attack [degrees].
Parameters	#aoa Freestream angle of attack [degrees].
Examples	–aoa 5

Table 8.47: BC alpha option

Beta	
Syntax	–beta #beta
Description	Override the freestream side slip angle [degrees].
Parameters	#beta Freestream side slip angle [degrees].
Examples	–beta 3

Table 8.48: BC beta option

U	
Syntax	–u #u
Description	Override the freestream x coordinate direction velocity [m/s].
Parameters	#u Freestream x coordinate direction velocity [m/s].
Examples	–u 150

Table 8.49: BC u option



V	
Syntax	<code>-v #v</code>
Description	Override the freestream y coordinate direction velocity [m/s].
Parameters	<code>#v</code> Freestream y coordinate direction velocity [m/s].
Examples	<code>-v 20</code>

Table 8.50: BC v option

W	
Syntax	<code>-w #w</code>
Description	Override the freestream z coordinate direction velocity [m/s].
Parameters	<code>#w</code> Freestream z coordinate direction velocity [m/s].
Examples	<code>-w 10</code>

Table 8.51: BC w option

T	
Syntax	<code>-T #T</code>
Description	Override the freestream temperature [K].
Parameters	<code>#T</code> Freestream temperature [K].
Examples	<code>-T 300</code>

Table 8.52: BC T option



p	
Syntax	<code>-p #p</code>
Description	Override the freestream pressure [Pa].
Parameters	<code>#p</code> Freestream pressure [Pa].
Examples	<code>-p 105000</code>

Table 8.53: BC p option

Trb.intensity	
Syntax	<code>-trb.intensity #ti</code>
Description	Override the freestream turbulent intensity (in absolute fraction, not percentage).
Parameters	<code>#ti</code> Freestream turbulent intensity (in absolute fraction, not percentage).
Examples	<code>-trb.intensity 0.01</code>

Table 8.54: BC trb.intensity option



8.11 System Options

A system refers to a “System of Equations,” *e.g.*, Euler or Reynolds Averaged Navier-Stokes equations. Other examples may include, turbulence model equations or various physical model equations (not included in the current revision of the code). Each system starts with the directive `--system`, followed by a series of options.

Certain systems require that an additional, complimentary system would be defined. For example, when simulating turbulent flows it is required to define a mean flow system that is consistent with the turbulence model selected and a second system to solve the actual turbulence model equations.

Type	
Syntax	<code>--type \$type</code>
Description	Type of equation set to be solved.
Parameters	<code>\$type</code> Type of equation set. See Table 8.56 for equation system types.
Examples	<code>--type ideal.gas.mf.inviscid</code> <code>--type ideal.gas.mf.tnt</code> <code>--type turbulent.kw.sst</code>

Table 8.55: System type option



System Types	
System type	Description
ideal.gas.mf.inviscid	Solve the Euler equations assuming inviscid, ideal gas flow.
ideal.gas.mf.viscous	Solve the Navier-Stokes equations assuming laminar ideal gas flow.
ideal.gas.mf.tnt	Solve the Navier-Stokes equations assuming turbulent ideal gas flow (using the $k - \omega$ -TNT turbulence model). This assumes an additional system for solving the $k - \omega$ -TNT turbulence model equations.
ideal.gas.mf.sst	Solve the Navier-Stokes equations assuming turbulent ideal gas flow (using the $k - \omega$ -SST turbulence model). This assumes an additional system for solving the $k - \omega$ -SST turbulence model equations.
turbulent.kw.tnt	Solve the $k - \omega$ -TNT turbulence model equations.
turbulent.kw.sst	Solve the $k - \omega$ -SST turbulence model equations.

Table 8.56: System types

Cell.gradient			
Syntax	<code>-cell.gradient \$method</code>		
Description	Method to calculate the cell center gradient that is required for high order approximations.		
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 30%;"><code>\$method</code></td> <td>Method for cell gradient calculation. See Table 8.58 for a list of available cell gradient methods.</td> </tr> </table>	<code>\$method</code>	Method for cell gradient calculation. See Table 8.58 for a list of available cell gradient methods.
<code>\$method</code>	Method for cell gradient calculation. See Table 8.58 for a list of available cell gradient methods.		
Examples	<code>-cell.gradient green.gauss.node</code>		

Table 8.57: System cell.gradient option



Cell Gradient Types	
Cell Gradient type	Description
green.gauss.node	Calculate cell gradient based on the Green-Gauss Theorem using node reconstruction (default).
green.gauss.cell	Calculate cell gradient based on the Green-Gauss Theorem using cell center reconstruction.
least.square.1	Calculate cell gradient based on least square approximations using r (the distance) as the weight.
least.square.1.5	Calculate cell gradient based on least square approximations using $r^{1.5}$ as the weight.
least.square.2	Calculate cell gradient based on least square approximations using r^2 as the weight.

Table 8.58: Cell gradient types

Face.gradient	
Syntax	–face.gradient \$method
Description	Method to calculate the face gradient that is required for viscous fluxes calculations.
Parameters	\$method Method for face gradient calculation. See Table 8.60 for a list of available face gradient methods.
Examples	–face.gradient diamond

Table 8.59: System face.gradient option



Face Gradient Types	
Face Gradient type	Description
diamond	Calculate face gradient based on the Green-Gauss Theorem using a diamond-like shape (default).
thin.layer	Calculate face gradient based on the thin layer assumption.
hasselbacher	Calculate face gradient based on defect correction by Hasselbacher.

Table 8.60: Face gradient types

Limiter	
Syntax	<code>–limiter \$limiter</code>
Description	Type of limiter.
Parameters	<code>\$limiter</code> Type of limiter. See Table 8.62 for limiter types.
Examples	<code>–limiter venka.2d</code>

Table 8.61: Limiter option



Limiter Types	
Limiter type	Description
none	No limiter.
1st	First order.
venka.2d	Venkatakrishnan limiter (2D domain).
venka.3d	Venkatakrishnan limiter (3D domain).
mlp.2d	MLP-u2 limiter (2D domain).
mlp.3d	MLP-u2 limiter (3D domain).

Table 8.62: Limiter types

Venka.k	
Syntax	<code>-venka.k #k</code>
Description	Change the Venkatakrishnan limiter coefficient. Default is 5.
Parameters	<code>#k</code> Venkatakrishnan limiter coefficient.
Examples	<code>-venka.k 5</code>

Table 8.63: System venka.k option



Time.step																			
Syntax	<code>–time.step #initial #iterations {cfl.linear cfl.exponential cfl.tangential dt.linear dt.exponential dt.tangential } #final</code>																		
Description	Sets the CFL number or time step (first order single time stepping).																		
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-right: 20px;"><code>#initial</code></td> <td>Initial CFL number or time step.</td> </tr> <tr> <td><code>#iterations</code></td> <td>Number of iteration for which the CFL number or time step grows from <code>#initial</code> to <code>#final</code>.</td> </tr> <tr> <td><code>cfl.linear</code></td> <td>Linear growth of the CFL number.</td> </tr> <tr> <td><code>cfl.exponential</code></td> <td>Exponential growth of the CFL number.</td> </tr> <tr> <td><code>cfl.tangential</code></td> <td>Hyperbolic tangent growth of the CFL number.</td> </tr> <tr> <td><code>dt.linear</code></td> <td>Linear growth of the time step.</td> </tr> <tr> <td><code>dt.exponential</code></td> <td>Exponential growth of the time step.</td> </tr> <tr> <td><code>dt.tangential</code></td> <td>Hyperbolic tangent growth of the time step.</td> </tr> <tr> <td><code>#final</code></td> <td>Final CFL number or time step.</td> </tr> </table>	<code>#initial</code>	Initial CFL number or time step.	<code>#iterations</code>	Number of iteration for which the CFL number or time step grows from <code>#initial</code> to <code>#final</code> .	<code>cfl.linear</code>	Linear growth of the CFL number.	<code>cfl.exponential</code>	Exponential growth of the CFL number.	<code>cfl.tangential</code>	Hyperbolic tangent growth of the CFL number.	<code>dt.linear</code>	Linear growth of the time step.	<code>dt.exponential</code>	Exponential growth of the time step.	<code>dt.tangential</code>	Hyperbolic tangent growth of the time step.	<code>#final</code>	Final CFL number or time step.
<code>#initial</code>	Initial CFL number or time step.																		
<code>#iterations</code>	Number of iteration for which the CFL number or time step grows from <code>#initial</code> to <code>#final</code> .																		
<code>cfl.linear</code>	Linear growth of the CFL number.																		
<code>cfl.exponential</code>	Exponential growth of the CFL number.																		
<code>cfl.tangential</code>	Hyperbolic tangent growth of the CFL number.																		
<code>dt.linear</code>	Linear growth of the time step.																		
<code>dt.exponential</code>	Exponential growth of the time step.																		
<code>dt.tangential</code>	Hyperbolic tangent growth of the time step.																		
<code>#final</code>	Final CFL number or time step.																		
Examples	<code>–time.step 5 50 cfl.linear 10</code>																		

Table 8.64: System CFL option



Implicit.jacobi	
Syntax	–implicit.jacobi
Description	Use Jacobi instead of the default Gauss-Seidel.
Parameters	N/A
Examples	–implicit.jacobi

Table 8.65: System implicit.jacobi option



Convection.noslip.diagonal							
Syntax	<code>–convection.noslip.diagonal {normal analytic implicit.on.noslip}</code>						
Description	Selects the way the diagonal convective jacobian is estimated on no-slip boundaries.						
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%; padding-right: 10px;">normal</td> <td>The inviscid diagonal is evaluated directly based on the scheme.</td> </tr> <tr> <td style="padding-right: 10px;">analytic</td> <td>Use analytical Jacobian</td> </tr> <tr> <td style="padding-right: 10px;">implicit.on.no</td> <td>Implicit treatment of wall boundary conditions through the Jacobian.</td> </tr> </table>	normal	The inviscid diagonal is evaluated directly based on the scheme.	analytic	Use analytical Jacobian	implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.
normal	The inviscid diagonal is evaluated directly based on the scheme.						
analytic	Use analytical Jacobian						
implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.						
Examples	<code>–convection.noslip.diagonal analytic</code>						

Table 8.66: System convection.noslip.diagonal option

Convection.impermeable.diagonal							
Syntax	<code>–convection.impermeable.diagonal {normal analytic implicit.on.noslip}</code>						
Description	Selects the way the diagonal convective jacobian is estimated on impermeable boundaries.						
Parameters	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%; padding-right: 10px;">normal</td> <td>The inviscid diagonal is evaluated directly based on the scheme.</td> </tr> <tr> <td style="padding-right: 10px;">analytic</td> <td>Use analytical Jacobian</td> </tr> <tr> <td style="padding-right: 10px;">implicit.on.no</td> <td>Implicit treatment of wall boundary conditions through the Jacobian.</td> </tr> </table>	normal	The inviscid diagonal is evaluated directly based on the scheme.	analytic	Use analytical Jacobian	implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.
normal	The inviscid diagonal is evaluated directly based on the scheme.						
analytic	Use analytical Jacobian						
implicit.on.no	Implicit treatment of wall boundary conditions through the Jacobian.						
Examples	<code>–convection.impermeable.diagonal analytic</code>						

Table 8.67: System convection.impermeable.diagonal option



Convection.symmetry.diagonal							
Syntax	<code>–convection.symmetry.diagonal {normal analytic implicit.on.noslip}</code>						
Description	Selects the way the diagonal convective jacobian is estimated on symmetry boundaries.						
Parameters	<table border="0"> <tr> <td><code>normal</code></td> <td>The inviscid diagonal is evaluated directly based on the scheme.</td> </tr> <tr> <td><code>analytic</code></td> <td>Use analytical Jacobian</td> </tr> <tr> <td><code>implicit.on.no</code></td> <td>Implicit treatment of wall boundary conditions through the Jacobian.</td> </tr> </table>	<code>normal</code>	The inviscid diagonal is evaluated directly based on the scheme.	<code>analytic</code>	Use analytical Jacobian	<code>implicit.on.no</code>	Implicit treatment of wall boundary conditions through the Jacobian.
<code>normal</code>	The inviscid diagonal is evaluated directly based on the scheme.						
<code>analytic</code>	Use analytical Jacobian						
<code>implicit.on.no</code>	Implicit treatment of wall boundary conditions through the Jacobian.						
Examples	<code>–convection.symmetry.diagonal analytic</code>						

Table 8.68: System `convection.symmetry.diagonal` option

Realizability.trb.w	
Syntax	<code>–realizability.trb.w</code>
Description	Adds realizability condition for omega (applies only to $k - \omega$ -SST turbulence model).
Parameters	N/A
Examples	<code>–realizability.trb.w</code>

Table 8.69: System `realizability.trb.w` option



Source.mpk		
Syntax	-source.mpk #mpk	
Description	Limit turbulence model production (applies only to $k-\omega$ turbulence models).	
Parameters	#mpk	Upper limit for production term.
Examples	-source.mpk 5	

Table 8.70: System source.mpk option

Damp		
Syntax	-damp #damp-factor	
Description	Set damping to the convection Jacobian. Values that are less than unity increase stability. Not recommended for turbulence model equations. The recommended damping factor is 0.5-1.0. Default is 1.	
Parameters	#damp-factor	Damping factor.
Examples	-damp 0.75	

Table 8.71: System damp option



Relax	
Syntax	<code>-relax #relax-factor</code>
Description	Set relaxation to the solution increment. The recommended relaxation factor is 0.5-1.0. Default is 1.
Parameters	<code>#relax-factor</code> Relaxation factor.
Examples	<code>-relax 0.75</code>

Table 8.72: System relaxation option

iteration.convergence	
Syntax	<code>-iteration.convergence #criterion</code>
Description	Set the convergence criterion ($\log(\text{Res}/\text{Res0})$), default is ignored. For unsteady flows this would be the convergence of the whole simulation. For dual time this would be convergence of the iterations.
Parameters	<code>#criterion</code> Convergence criterion in terms of ($\log(\text{Res}/\text{Res0})$).
Examples	<code>-iteration.convergence -5</code>

Table 8.73: System iteration.convergence option



Log	
Syntax	<code>-log \$log [log-options] ...</code>
Description	Report in log file. See Table 8.75 for log options list.
Parameters	<code>\$log</code> Log name.
Examples	<code>-log coefficients cl cd</code>

Table 8.74: System log option



System Log Options	
Log type	Description
cfl	System CFL.
dt	System Δt .
time	Time.
residual	System residual.
log.residual	System log of the residual.
minimal.timestep	Minimal time step.
orig(minimal.timestep)	Minimal time step original cell index.
cell(minimal.timestep)	Minimal time step zonal cell index.
zone(minimal.timestep)	Minimal time step zone index.
orig(res)	Max residual original cell number.
cell(res)	Max residual zonal cell number.
zone(res)	Max residual zone number.
max(mt)	Max of the turbulence viscosity (where applicable).
cell(mt)	Max turbulent viscosity cell number (where applicable).
zone(mt)	Max turbulent viscosity zone number (where applicable).

Table 8.75: System log options



Plot	
Syntax	<code>-plot \$plot [plot-options] ...</code>
Description	Functions to include in FV-UNS file. See Table 8.77 for plot functions list.
Parameters	<code>\$plot</code> Plot name.
Examples	<code>-plot NACA0012 r p velocity residual</code>

Table 8.76: System plot option



Plot Functions	
Plot functions	Description
residual	Residual field.
relax	Relaxation field.
velocity	Velocity vector field (where applicable).
{u v w }	Velocity vector field components (scalars, where applicable).
r	Density field (where applicable).
p	Pressure field (where applicable).
Cp	Pressure coefficient (where applicable).
T	Temperature field (where applicable).
{lim.r lim.p lim.u lim.v lim.w }	Limiter fields (where applicable).
wall.distance	Wall distance field (where applicable).
mt	Turbulent viscosity field (where applicable).
trb.k	Turbulent kinetic energy field (where applicable).
trb.w	Turbulent specific dissipation rate field (where applicable).
{trb.lim.k trb.lim.w }	Turbulence model limiters (where applicable).
Tw	Magnitude of $\bar{\tau}_{wall}$ (where applicable).
Tw.vec	The vector $\bar{\tau}_{wall}$ (where applicable).
Tw{x y z }	X Y Z direction component of $\bar{\tau}_{wall}$.

Table 8.77: Plot functions

8.12 Solve Options

The solve input options section starts with the `--solve` directive, followed by a series of solve options.

Convection.flux	
Syntax	<code>--convection.flux \$method</code>
Description	Set the convection flux approximation method.
Parameters	<code>\$method</code> Flux approximation method. See Table 8.79 for available convection flux types.
Examples	<code>--convection.flux hllc.roe</code>

Table 8.78: Solve convection.flux option

Convection Flux Types	
Convection flux type	Description
<code>hllc.roe</code>	HLLC Roe.
<code>hllc.davis</code>	HLLC Davis.
<code>ausm</code>	AUSM.
<code>aufsr</code>	AUFSR.
<code>aufsr+</code>	AUFSR+ (recommended for low Mach number flow simulations).

Table 8.79: Convection flux types



Convection.jacobian	
Syntax	<code>–convection.jacobian \$jacobian</code>
Description	Set the convection Jacobian. See Table 8.81 for available convection Jacobian types.
Parameters	<code>\$jacobian</code> Jacobian.
Examples	<code>–convection.jacobian hllc.roe</code>

Table 8.80: Solve convection.jacobian option

Convection Jacobian Types	
Convection Jacobian type	Description
<code>hllc.roe</code>	HLLC Roe.
<code>hllc.davis</code>	HLLC Davis.
<code>van.leer</code>	van Leer.
<code>rossow.low.mach</code>	Low Mach number Rossow.
<code>matrix.free</code>	Matrix free.

Table 8.81: Convection Jacobian types



Spatial.order	
Syntax	<code>–spatial.order #</code>
Description	Set the convection flux spatial order. Available options are either 1 for first order or 2 for second order.
Parameters	<code>#</code> Convection flux approximation order.
Examples	<code>–spatial.order 2</code>

Table 8.82: Solve spatial.order option

Sweeps	
Syntax	<code>–sweeps #</code>
Description	Set the number of sweeps within an iteration.
Parameters	<code>#</code> Number of sweeps. Can be any even number. Default is 6.
Examples	<code>–sweeps 4</code>

Table 8.83: Solve sweeps option



Time.march	
Syntax	<code>–time.march \$method</code>
Description	Set the time marching method. See Table 8.85 for a list of available time marching methods.
Parameters	<code>\$method</code> Time marching method.
Examples	<code>–time.march implicit.pgs</code>

Table 8.84: Solve time.march option



Time Marching Methods

Time marching method	Description
explicit.euler	Explicit Euler time marching scheme.
explicit.rk3	Third order Runge-Kutta explicit time marching scheme.
explicit.rk4	Fourth order Runge-Kutta explicit time marching scheme.
implicit.pgs	Implicit point Gauss-Seidel time marching scheme.
implicit.b2.pgs	Second order implicit B2 time marching scheme.
implicit.heun.R.pgs	Second order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk3.R.pgs	Third order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk4.R.pgs	Fourth order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk5.R.pgs	Fifth order Runge-Kutta implicit point Gauss-Seidel time marching scheme.
implicit.rk3.R.petsc	Krylov-based Implicit (Runge-Kutta R, 3rd order).
implicit.rk4.R.petsc	Krylov-based Implicit (Runge-Kutta R, 4th order).
implicit.b2.petsc	Krylov-based Implicit (B2).

Table 8.85: Time marching methods

Iterations	
Syntax	<code>-iterations #</code>
Description	Set the number of iterations. For unsteady flows this would be the number of iterations of the whole simulation. For dual time this would be the maximum iterations of the current time step.
Parameters	<code>#</code> Number of iterations.
Examples	<code>-iterations 1000</code>

Table 8.86: Solve iterations option

Time.step.reduction	
Syntax	<code>-time.step.reduction #attempts</code>
Description	Set time the number of consecutive CFL reduction attempts when non-physical values are detected in the solution.
Parameters	<code>#attempts</code> Number of attempts.
Examples	<code>-time.step.reduction 3</code>

Table 8.87: Solve time.step.reduction option



Dual.time					
Syntax	<code>-dual.time #steps #time-step</code>				
Description	Set dual time stepping simulation.				
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><code>#steps</code></td> <td>Number of physical time steps.</td> </tr> <tr> <td><code>#time-step</code></td> <td>Physical time step (Δt).</td> </tr> </table>	<code>#steps</code>	Number of physical time steps.	<code>#time-step</code>	Physical time step (Δt).
<code>#steps</code>	Number of physical time steps.				
<code>#time-step</code>	Physical time step (Δt).				
Examples	<code>-dual.time 200 0.0001</code>				

Table 8.88: Solve dual.time option

Global.minimal.timestep	
Syntax	<code>-global.minimal.timestep</code>
Description	Forces all the cells to use the global minimal time step that was calculated.
Parameters	N/A
Examples	<code>-global.minimal.timestep</code>

Table 8.89: Solve global.minimal.timestep option

Save			
Syntax	<code>-save #</code>		
Description	Set the intermittency of output of restart files.		
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 15%;"><code>#</code></td> <td>Number of steps between output.</td> </tr> </table>	<code>#</code>	Number of steps between output.
<code>#</code>	Number of steps between output.		
Examples	<code>-save 10</code>		

Table 8.90: Solve save option



Save.path	
Syntax	<code>–save.path \$path</code>
Description	Set the path for output of log and restart files.
Parameters	<code>\$path</code> Path of log and restart files.
Examples	<code>–save.path ./save/</code>

Table 8.91: Solve save.path option

Save.sequential	
Syntax	<code>–save.sequential</code>
Description	Adds iteration signature to the saves.
Parameters	N/A
Examples	<code>–save.sequential</code>

Table 8.92: Solve save.sequential option

Load.path	
Syntax	<code>–load.path \$path</code>
Description	Set the path for restart files.
Parameters	<code>\$path</code> Path of restart files.
Examples	<code>–load.path ./load/</code>

Table 8.93: Solve load.path option



Load.sequence	
Syntax	<code>-load.sequence #iteration</code>
Description	Load a specific iteration.
Parameters	<code>#iteration</code> Iteration for load.
Examples	<code>-load.sequence 1000</code>

Table 8.94: Solve load.sequence option

EOS	
Syntax	<code>-eos \$eos</code>
Description	Set the equation of state.
Parameters	<code>\$eos</code> Equation of state. Currently, only perfect gases are supported.
Examples	<code>-eos air</code>

Table 8.95: Solve eos option

Molecular	
Syntax	<code>-molecular \$molecular</code>
Description	Set the name of the molecular viscosity and thermal conductivity to use.
Parameters	<code>\$molecular</code> Molecular relations name.
Examples	<code>-molecular air-suther</code>

Table 8.96: Solve molecular option



P	
Syntax	<code>-p #</code>
Description	Set the free stream pressure.
Parameters	# Free stream pressure.
Examples	<code>-p 101325</code>

Table 8.97: Solve p option

T	
Syntax	<code>-T #</code>
Description	Set the free stream temperature.
Parameters	# Free stream temperature.
Examples	<code>-p 288</code>

Table 8.98: Solve T option

Mach	
Syntax	<code>-Mach #</code>
Description	Set the free stream Mach number.
Parameters	# Free stream Mach number.
Examples	<code>-Mach 0.8</code>

Table 8.99: Solve Mach option



Velocity.magnitude	
Syntax	<code>-velocity.magnitude #</code>
Description	Set the free stream velocity magnitude.
Parameters	<code>#</code> Free stream velocity magnitude.
Examples	<code>-velocity.magnitude 200</code>

Table 8.100: Solve velocity magnitude option

U	
Syntax	<code>-u #</code>
Description	Set the free stream X coordinate direction velocity component.
Parameters	<code>#</code> Free stream X coordinate direction velocity component.
Examples	<code>-u 200</code>

Table 8.101: Solve u option

V	
Syntax	<code>-v #</code>
Description	Set the free stream Y coordinate direction velocity component.
Parameters	<code>#</code> Free stream Y coordinate direction velocity component.
Examples	<code>-v 0</code>

Table 8.102: Solve v option



W	
Syntax	<code>-w #</code>
Description	Set the free stream Z coordinate direction velocity component.
Parameters	<code>#</code> Free stream Z coordinate direction velocity component.
Examples	<code>-w 20</code>

Table 8.103: Solve w option

Alpha	
Syntax	<code>-alpha #</code>
Description	Set the free stream angle of attack.
Parameters	<code>#</code> Free stream angle of attack in degrees.
Examples	<code>-alpha 10</code>

Table 8.104: Solve alpha option

Beta	
Syntax	<code>-beta #</code>
Description	Set the free stream side slip angle.
Parameters	<code>#</code> Free stream side slip angle in degrees.
Examples	<code>-beta 0</code>

Table 8.105: Solve beta option



Trb.intensity		
Syntax	-trb.intensity #	
Description	Set the free stream turbulence intensity.	
Parameters	#	Free stream turbulence intensity (in absolute fraction, not percentage).
Examples	-trb.intensity 0.01	

Table 8.106: Solve trb.intensity option

Trb.mt		
Syntax	-trb.mt #mt	
Description	Set the free stream turbulence viscosity.	
Parameters	#mt	Free stream turbulence viscosity (in absolute fraction from the molecular viscosity, not percentage).
Examples	-trb.mt 0.01	

Table 8.107: Solve trb.mt option



Reference.mach.trb	
Syntax	<code>–reference.mach.trb #ref</code>
Description	Set the reference Mach number for the turbulence model (used to override the freestream Mach number).
Parameters	<code>#ref</code> Freestream Mach number.
Examples	<code>–reference.mach.trb 0.7</code>

Table 8.108: Solve reference.mach.trb option

Reference.velocity.trb	
Syntax	<code>–reference.velocity.trb #ref</code>
Description	Set the reference velocity for the turbulence model (used to override the freestream velocity).
Parameters	<code>#ref</code> Freestream velocity [m/s].
Examples	<code>–reference.velocity.trb 210</code>

Table 8.109: Solve reference.velocity.trb option

Reference.mach	
Syntax	<code>–reference.mach #ref</code>
Description	Set the reference Mach number. Default is the mean flow free stream Mach number.
Parameters	<code>#ref</code> Reference Mach number.
Examples	<code>–reference.mach 0.75</code>

Table 8.110: Solve reference.mach option



Reference.velocity	
Syntax	<code>-reference.velocity #ref</code>
Description	Set the reference velocity. Default is the mean flow free stream velocity.
Parameters	<code>#ref</code> Reference velocity.
Examples	<code>-reference.velocity 300</code>

Table 8.111: Solve reference.velocity option

Reference.longitudinal	
Syntax	<code>-reference.longitudinal #ref</code>
Description	Set the reference length for Reynolds number evaluation. Unused by the current version.
Parameters	<code>#ref</code> Reference length.
Examples	<code>-reference.length 1.5</code>

Table 8.112: Solve reference.length option



Ref									
Syntax	<code>-ref.\$ # { # # }</code>								
Description	Set the reference length, area, and reference point for force and moment coefficient calculations.								
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="width: 30%;"><code>-reference.area</code></td> <td>Reference area.</td> </tr> <tr> <td><code>-reference.len</code></td> <td>Reference length.</td> </tr> <tr> <td><code>-reference.point</code></td> <td>Reference point.</td> </tr> <tr> <td><code>#</code></td> <td>Reference value.</td> </tr> </table>	<code>-reference.area</code>	Reference area.	<code>-reference.len</code>	Reference length.	<code>-reference.point</code>	Reference point.	<code>#</code>	Reference value.
<code>-reference.area</code>	Reference area.								
<code>-reference.len</code>	Reference length.								
<code>-reference.point</code>	Reference point.								
<code>#</code>	Reference value.								
Examples	<pre> -reference.len 0.325 -reference.point 1 0 0 </pre>								

Table 8.113: Solve ref option

Plot	
Syntax	<code>-plot [plot-options] ...</code>
Description	Add to the fvuns file. See Table 8.115 for plot options list.
Parameters	N/A
Examples	<code>-plot rank owner orig</code>

Table 8.114: Solve plot option



Solve Plot Options	
Log type	Description
rank	Add the rank to the fvuns plot file.
part	Add the part to the fvuns plot file.
owner	Add ownership to the fvuns plot file.
orig	Add original node index to the fvuns plot file.

Table 8.115: Solve plot options

Log	
Syntax	<code>–log \$log [log-options] ...</code>
Description	Report in log file. See Table 8.117 for log options list.
Parameters	<code>\$log</code> Log name.
Examples	<code>–log progress-logs iter step time</code>

Table 8.116: Solve log option



Solve Log Options	
Log type	Description
iter	Iteration number.
step	Step number.
pos	Position # (iteration for steady state, step for dual time simulation).
time	Physical time for dual time simulation.
exec.time	Cumulative execution time.

Table 8.117: Solve log options



8.13 PETSc Toolkit

PETSc, the Portable, Extensible Toolkit for Scientific Computation is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. PETSc is developed as open-source.

The PETSc input option section starts with the `--petsc` directive, followed by a series of petsc options. For advanced usage of the PETSc package the user is referred to the command line help or the [online PETSc manual](#).

8.14 UNS Options

The UNS input options section start with the `--uns` directive, followed by a series of uns options.

Name	
Syntax	<code>--name \$name</code>
Description	Name of flow case.
Parameters	<code>\$name</code> Flow case name.
Examples	<code>--name naca0012_laminar</code>

Table 8.118: UNS name option



Prefix	
Syntax	<code>–prefix \$prefix</code>
Description	Set prefix of the flow case.
Parameters	<code>\$prefix</code> Prefix of flow case.
Examples	<code>–prefix ./naca0012/NACA0012-Str-Laminar-Rey500</code>

Table 8.119: UNS prefix option

Key	
Syntax	<code>–key \$prefix</code>
Description	Save/load a key file. The file contains pre-calculated wall distance. If the file exists, the code reads it prior to execution. If the file does not exist, the code generates one.
Parameters	<code>\$prefix</code> Prefix of the key file.
Examples	<code>–key ./NACA0012-Str-Laminar-Rey500</code>

Table 8.120: UNS key option



Scale		
Syntax	-scale #x #y #z	
Description	Scale the grid.	
Parameters	#x	Scale in the x direction.
	#y	Scale in the y direction.
	#z	Scale in the z direction.
Examples	-scale 0.001 0.001 0.001	

Table 8.121: UNS scale option

Log		
Syntax	-log \$log [log-options] ...	
Description	Report in log file. See Table 8.123 for log options list.	
Parameters	\$log	Log name.
Examples	-log coefficients cl cd	

Table 8.122: UNS log option



UNS Log Options	
Log type	Description
wet.surface	Wet surface area.
fx	X coordinate direction force.
fy	Y coordinate direction force.
fz	Z coordinate direction force.
lift	Lift force.
drag	Drag force.
mx	Moment about X coordinate direction.
my	Moment about Y coordinate direction.
mz	Moment about Z coordinate direction.
cfx	X coordinate direction force coefficient.
cfy	Y coordinate direction force coefficient.
cfz	Z coordinate direction force coefficient.
cmx	Moment coefficient about X coordinate direction.
cmx	Moment coefficient about Y coordinate direction.
cmz	Moment coefficient about Z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.

Table 8.123: UNS log options



8.15 Run-time Options

Arion provides the capability to control the run while the application is running using the run-time options. This can be achieved by creating an ascii text file named ‘arion.ins’ containing one of the following run-time options:

Run-time Options	
Option	Action
stop	Stop and save a restart.
kill	Immediately stop the run (even in dual-time mode) and save a restart.
plot	Dump the FVUNS file as requested by the user at the next available opportunity.
save	Save a restart.
time.step [\$system] #cfl	Change to the specified constant CFL or virtual time step of all the systems or the specified \$system.
dt #dt	Change to the specified physical time step (for dual time simulations).

Table 8.124: Run-time options

8.15.1 Run-time Options Examples

The examples in Table 8.125 provide the complete Unix/Linux command to create the ‘arion.ins’ file with the required content. Alternatively, the user may use an editor to create the file and enter the content. Note, once **Arion** identifies the existence of the file ‘arion.ins’ and reads its content, the file is erased.



Run-time Options Examples	
Command	Result
echo stop > arion.ins	Stop and save a restart.
echo plot > arion.ins	Dump the FVUNS file as requested by the user at the next available opportunity.
echo save > arion.ins	Save a restart.
echo time.step 50.0 > arion.ins	Change the CFL or time step to 50.0 (all systems).
echo time.step turbulent.kw.tnt 30.0 > arion.ins	Change the CFL or virtual time step for the k- ω -TNT turbulence model equations to 30.0.
echo dt 0.001	Change the physical time step to 0.001

Table 8.125: Run-time options examples



Bibliography

- [1] F. R. Menter. Zonal two equation $k-\omega$ turbulence models for aerodynamic flows. In *24rd AIAA Fluid Dynamics Conference*, Orlando, FL, July 1993. AIAA paper 93 - 2906.
- [2] Ami Harten, Peter D. Lax, and Bram Van Leer. On upstream differencing and godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):35–61, 1983.
- [3] E. F. Toro, M. Spruce, and W. Spears. Restoration of the contact surface in the hll riemann solver. *Shock Waves*, 4(4):25–34, 1994.
- [4] P. Batten, M. A. Leschziner, and U. C. Goldberg. Average-state Jacobians and implicit methods for compressible viscous and turbulent flows. *Journal of Computational Physics*, 137(1):38–78, 1997.
- [5] B. Einfeldt, C. D. Munz, P. L. Roe, and B. Sjogreen. On Godunov-type methods near low densities. *Journal of Computational Physics*, 92(2):273–295, 1991.
- [6] P. L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.
- [7] SF Davis. Simplified second-order godunov-type methods. *SIAM Journal on Scientific and Statistical Computing*, 9(3):445–473, 1988.
- [8] Meng-Sing Liou and Christopher J. Steffen Jr. A new flux splitting scheme. *Journal of Computational Physics*, 107(1):23–39, 1993.

- [9] Meng-Sing Liou. A sequel to AUSM: $AUSM^+ - up$. *Journal of Computational Physics*, 129:364–382, 1996.
- [10] Meng-Sing Liou. A sequel to AUSM, part II: $AUSM^+ - up$ for all speeds. *Journal of Computational Physics*, 214:137–170, 2006.

