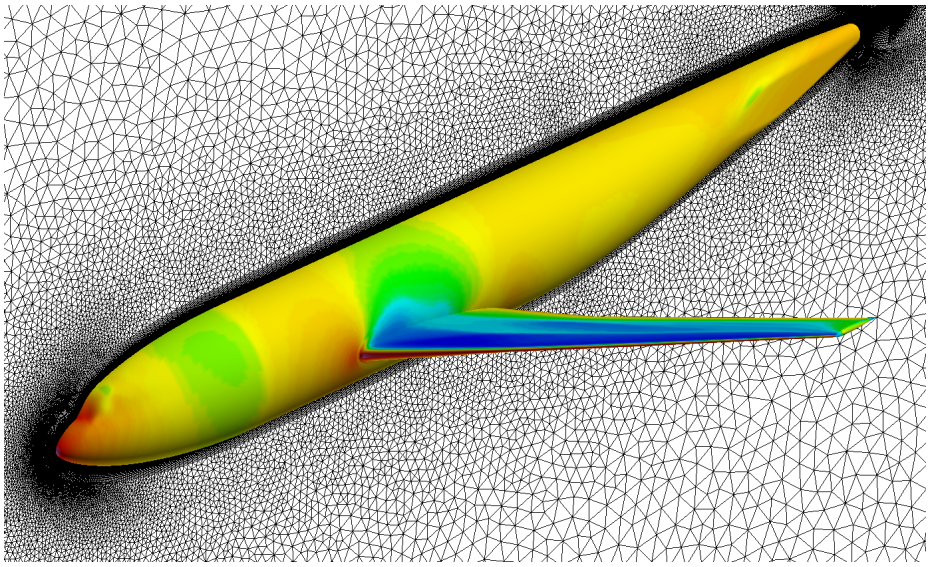
THE ARION UNSTRUCTURED GRIDS CFD SOLVER: THEORETICAL AND USER'S MANUAL

Version 1.23 (Revision 553)

ISCFDC Report 2017-07



Prepared by

Ya'eer Kidron, Yair Mor-Yossef, and Yuval Levy

Contents

List of Figures

List of Tables

Listings

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, or the AUSM$^+$-up scheme for the approximation of the convective fluxes and the k$-\omega$-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 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, or the AUSM^{+}-up scheme for the approximation of the convective fluxes and the k$-\omega$-TNT turbulence model. The code is fully parallel using a distributed architecture, as well as by shared memory (OpenMP) using hybrid architecture.

1.2 Near Future Releases

It is planned to release a new version of the code every 1-2 months during the remainder of the year 2016 and the year 2017. Each of the version shall contain new feature as well as bug fixes. Each version shall be accompanied by an updated Users manual as well as validation report.

1.3 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 supports. 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

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

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

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \tag{2.2}$$

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

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

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

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

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 $\overline{V}_g$, the integral form of the conservation equations in an inertial frame of reference is given as:

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

where

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

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:

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

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

2.2.3 Thermodynamic Properties

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:

$$p = \rho\,(\gamma - 1)\,e_I \tag{2.10}$$

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:

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

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 relations for the coefficients of viscosity, μ, and thermal conductivity, κ. In the single-component perfect gas (SPG) model, the Sutherland formulae are exclusively used to evaluate these coefficients as follows:

$$\begin{aligned} \mu &= C_{\mu 1}\frac{T^{\frac{3}{2}}}{T + C_{\mu 2}} \\ \kappa &= C_{\kappa 1}\frac{T^{\frac{3}{2}}}{T + C_{\kappa 2}} \end{aligned} \tag{2.12}$$

The default of the coefficients $C_{\mu 1}$, $C_{\mu 2}$, $C_{\kappa 1}$, and $C_{\kappa 1}$ correspond to air and are given in Table 2.1. The coefficients can be set by the user using the directives as described in Tables 8.25, 8.26, 8.27, 8.28.

Fluid type	$C_{\mu 1}$	$C_{\mu 2}$	$C_{\kappa 1}$	$C_{\kappa 1}$
Air	1.458×10^{-6}	110.4	2.495×10^{-3}	194

Table 2.1: Default coefficients for Sutherland formulae

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 8.23), the the heat capacity ratio, (γ, see Table 8.24, 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 know 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):

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

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 $(\bar{\ })$ indicates non-weighted averaging, the symbol $(\tilde{\ })$ signifies mass weighted Favre averaging, and the symbol $(\ '')$ 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:

$$Q = \begin{bmatrix} \bar{\rho} \\ \bar{\rho}\tilde{u} \\ \bar{\rho}\tilde{v} \\ \bar{\rho}\tilde{w} \\ \tilde{E} \\ \bar{\rho}q_1 \\ \dots \\ \bar{\rho}q_m \end{bmatrix}, \qquad S = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ s1 \\ \dots \\ s_m \end{bmatrix} \tag{3.2}$$

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:

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

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

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

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

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

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:

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

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

$$\mu_t = \frac{\overline{\rho} k}{\omega} \tag{3.8}$$

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 3.3.

3.3 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:

$$k_{wall} = 0 \tag{3.9}$$

$$\omega_{wall} = 10 \frac{6\nu}{\beta_1 \left(\Delta d_1\right)^2} \tag{3.10}$$

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

$$k_\infty = \frac{3}{2} \left(T_u \cdot U_\infty\right)^2 \tag{3.11}$$

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:

$$\omega_\infty = \frac{\bar{\rho}_\infty k_\infty}{\left(\mu_t\right)_\infty} \tag{3.12}$$

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

$$0.01 < \frac{(\mu_t)_\infty}{(\mu)_\infty} < 1.0 \tag{3.13}$$

3.4 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:

$$\Re_{ij} = \Re_{ij}^{LEVM} \tag{3.14}$$

where

$$\Re_{ij}^{LEVM} = 2\mu_t \left(S_{ij} - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij} \right) - \frac{2}{3}\bar{\rho}k\delta_{ij} \tag{3.15}$$

and

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3.16}$$

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.

3.5 Turbulence-Mean-flow Coupling

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

$$\begin{aligned} \mu &= \bar{\mu}_l + \bar{\mu}_t \\ \kappa &= \bar{\kappa}_l + \frac{C_p\bar{\mu}_t}{Pr_t} \end{aligned} \tag{3.17}$$

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:

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

where H is the rotated flux, namely,

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

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:

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

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 [1] 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* [2] discussed this limitation, and proposed a modified three wave

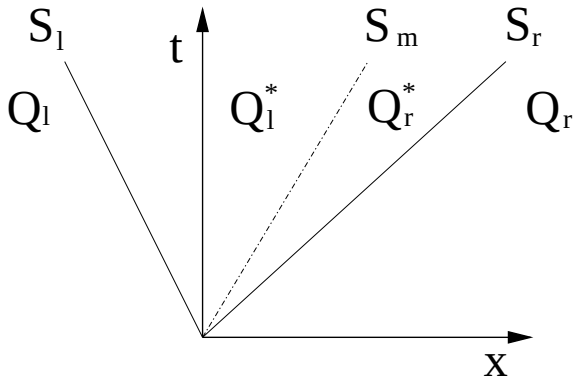


Figure 4.1: Wave structure of the HLLC approximate Riemann solver

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 [3].

The HLLC approximate Riemann solver that is implemented in the **Arion** code is as proposed by Batten *et al* [3]. 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* [4] as follows:

$$S_l = \min[\lambda_{min}, \lambda_{min}^{Roe}] \tag{4.4}$$

$$S_r = \max[\lambda_{max}, \lambda_{max}^{Roe}] \tag{4.5}$$

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 [5], respectively. In the second, the wave speeds S_l and S_r are computed according to Davis [6] as follows:

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

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

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

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

The contact discontinuity speed S_m is evaluated according to Batten *et al* [3] by

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

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

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

Introducing the intermediate left state vector

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

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

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

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

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

where $\mathbf{H}_c\left(\mathbf{Q}_l\right)$ and $\mathbf{H}_c\left(\mathbf{Q}_r\right)$ 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 [7]. 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$^+$ [8]. 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 [9] extended for all speed flows. The AUSM$^+$-up is implemented in the **Arion** code and it is given as follows:

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

where

$$\boldsymbol{\psi}_{l/r} = \begin{Bmatrix} 1 \\ u_{l/r} \\ v_{l/r} \\ w_{l/r} \\ H_{l/r} \end{Bmatrix}, \tag{4.15}$$

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

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

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

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

where

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

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:

$$M_{l/r} = \frac{q_{l/r}}{a_{1/2}} \tag{4.19}$$

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:

$$a_{1/2} = \frac{a_l + a_r}{2} \tag{4.20}$$

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

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

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:

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

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

$$\mathcal{P}_{(5)}^{\pm}(M) = \begin{Bmatrix} \frac{1}{M}\mathcal{M}_{(1)}^{\pm} & if\ |M| \geq 1 \\ \mathcal{M}_{(2)}^{\pm}[(\pm 2 - M) \mp 16\alpha M\mathcal{M}_{(2)}^{\mp}) & otherwise \end{Bmatrix} \tag{4.23}$$

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 [3] 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:

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

The HLLC inviscid flux is then easily evaluated using:

$$\mathbf{H}_{cl}^{*} \equiv \mathbf{H}_{c}\left(\mathbf{Q}_{l}^{*}\right) = \mathbf{Q}_{l}^{*} S_{m} + \left\{ \begin{array}{c} 0 \\ p^{*} n_{x} \\ p^{*} n_{y} \\ p^{*} n_{z} \\ p^{*} S_{m} \\ 0 \\ 0 \end{array} \right\} \tag{4.25}$$

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 $\boldsymbol{W} = (W_m; m = 1, ...7)$ denote the primitive variables vector (for a general $k - \omega$ turbulence model),

$$\boldsymbol{W} = [\bar{\rho}, \widetilde{u}, \widetilde{v}, \widetilde{w}, \bar{p}, k, \omega] \tag{4.26}$$

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

$$\left(W_m\right)_L = \left(W_m\right)_i + \left(\psi_m\right)_i \left(\nabla W_m\right)_i \cdot \boldsymbol{d}_i^{ij} \tag{4.27a}$$

$$\left(W_m\right)_R = \left(W_m\right)_j + \left(\psi_m\right)_j \left(\nabla W_m\right)_j \cdot \boldsymbol{d}_j^{ij} \tag{4.27b}$$

where $\boldsymbol{d}_i^{ij}$ $\left(\boldsymbol{d}_j^{ij}\right)$ 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 equation[1]:

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

[1] Equation (4.3) without the index

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

$$\Delta Q^n = \frac{\Delta t}{V} R^n \tag{4.29}$$

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

$$\Delta Q^n = Q^{n+1} - Q^n \tag{4.30}$$

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:

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

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:

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

where R^n is the residual at time level n as define 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:

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

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:

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

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

5.2.2 No-Slip Condition

The no-slip condition is much easier to implement:

$$\overline{V}_g = 2\overline{V}_f - \overline{V}_r \tag{5.2}$$

5.2.3 Adiabatic Wall

The temperature is set using:

$$T_g = T_r \tag{5.3}$$

while the pressure is set using

$$p_g = p_r \tag{5.4}$$

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

$$\rho_g = \rho_\infty \tag{5.5}$$
$$\overline{V}_g = \overline{V}_\infty \tag{5.6}$$
$$p_g = \frac{\rho_\infty}{\rho_r} p_r \tag{5.7}$$

5.3.1.2 Turkel Outlet

$$\rho_g = \rho_r + \frac{p_\infty - p}{a_\infty^2} \tag{5.8}$$
$$\overline{V}_g = \overline{V}_r \tag{5.9}$$
$$p_g = p_\infty \tag{5.10}$$

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:

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

The ghost Riemann invariants are obtained using:

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

5.3.2.1 Riemann Inlet

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

5.3.2.2 Riemann Outlet

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

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

5.3.3 Fixed (Supersonic Inlet)

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

5.3.4 Extrapolation (Supersonic Outlet)

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

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.

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

```
--parallel
   -cache 100000
   -rank 1 -threads 8

--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
   -type ideal.gas.mf.inviscid
   -time.step 5 100 cfl.exponential 500
   -cell.gradient green.gauss.node
   -limiter mlp.2d

   !logs
```

```
   -log convergence cfl residual log.residual

   !plots
   -plot NACA0012-Jameson-129 p velocity r residual

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

   -iterations 1500
   -save        100
   -save.path ./save/

   -log convergence iter

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

--log
   -name convergence
   -prefix ./logs/convergence
   -per pos
```

```
--plot
   -name NACA0012-Jameson-129
   -prefix ./plot/
   -interval 100

--uns
   -name Jameson_naca0012_129
   -prefix ./naca0012-j129-str/naca0012-j-129-arion-4-1-2
   -min.parts 8
   -max.parts 8
```

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. The following section describes the 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. The following section contains the types of file conversions and the instructions for the conversion.

8.3 Conversion Options

As mentioned in the previous section, for higher efficiency, the Star-CD input files are converted to the **Arion** format with various internal reordering possibilities. The

following types are available: *bin*, *snake*, *split*, *level*, and *arion*. Starting from version 1.19 the user must convert the files prior to running the code. This is conducted by invoking the **Arion** code using the possibilities that are outlines in Table 8.1.

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 various conversion types differ from one another by the splitting and/or reordering of the grid information. Thanks to the reordering and splitting, the grid files are read in parallel by the **Arion** code, accelerating further the read process. 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	arion −−star2{bin | snake | split | level | arion} $prefix {null \| null \| #x # y #z \| #levels \| #x # y #z} [$output]	
Description	Conversion of the Star-CD files. Note that star2bin and star2snake require no additional arguments.	
Parameters	−−star2bin	Converts Star-CD files to binary format.
	−−star2snake	Similar to star2bin but reorders the cells in a snake like order.
	−−star2split #x #y #z	Converts and splits the Star-CD files to binary format and reorders the cells in a geometric like order. The symbols #x, #y, #z signify the number of divisions in the respective direction.
	−−star2level #levels	Converts Star-CD files to binary format and reorders the cells in a geometric like order. The symbol #levels, signifies the number of levels.
	−−star2arion #x #y #z	Similar to star2split but reorders the cells in a snake like order.
	$prefix	Prefix of Star-CD files.
	$output	Prefix of output files.
Examples	arion −−star2arion Star-CD-file-name 8 8 8 New-file-name	
	arion −−star2snake Star-CD-file-name	

Table 8.1: Conversion options

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 8.2.

cnv2cnv		
Syntax	arion −−cnv2cnv $cnv1 $cnv2 $prefix2 {$system1-prefix}...	
Description	Converts **Arion** solution files between different orderings; creates new restart files.	
Parameters	$cnv1	Ordering type to convert from.
	$cnv2	Ordering type to convert to.
	$prefix2	Prefix of output files
	$system1-prefix	Prefix of converted system
Examples	arion −−cnv2cnv bin arion ./newsolution/newfiles ./restart/oldfiles	

Table 8.2: 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	−name $name
Description	Sets the name of the log. This name is referred to by the specific −log 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	$name Log name.
Examples	−name convergence

Table 8.3: Log name option

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

Table 8.4: Log prefix option

Per	
Syntax	−per $type
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	$type Type (iter, step, or pos).
Examples	−per pos

Table 8.5: 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 N/A
Examples	−screen

Table 8.6: Log stdout option

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

Table 8.7: 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	−name $name	
Description	Sets the name of the plot. This name is referred to by the specific −plot options.	
Parameters	$name	Plot name.
Examples	−name Alpha10	

Table 8.8: Plot name option

Prefix		
Syntax	−prefix $prefix	
Description	The plot file path and name prefix. The actual file number is composed from the prefix, and the $name as described in Table 8.93. The '.fvuns' 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	$prefix	Prefix.
Examples	−prefix ./plots/	

Table 8.9: Plot prefix option

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

Table 8.10: Plot interval option

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

Table 8.11: Plot sequential/overwrite 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	−name $name
Description	Sets the name of the table. This name is referred to by the specific −table options.
Parameters	$name Table name.
Examples	−name Alpha10

Table 8.12: Table name option

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

Table 8.13: Table prefix option

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

Table 8.14: Table interval option

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

Table 8.15: Table sequential/overwrite option

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

Table 8.16: Table horizontal/vertical option

Ray		
Syntax	−ray #x #y #z #vx #vy #vz	
Description	Create a table along a predefined ray.	
Parameters	#x #y #z	Ray origin.
	#vx #vy #vz	Ray direction.
Examples	−ray 0 0 0 1 1 1	

Table 8.17: Table ray option

Plane.bc		
Syntax	−plane.bc #x #y #z #nx #ny #nz $bc-name	
Description	Create a table along an intersection between a plane and a boundary surface.	
Parameters	#x #y #z	Plane.bc origin.
	#nx #ny #nz	Plane.bc direction.
	$bc-name	The boundary surface name as assigned by the −−bc −name directive (see Table 8.30).
Examples	−plane.bc 0 0 0 1 1 1 WALL	

Table 8.18: 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	−cache #	
Description	Size of the cache memory (in bytes).	
Parameters	#	Cache size.
Examples	−cache 1000	

Table 8.19: Parallel cache option

Rank		
Syntax	[{-rank #rank | -all.ranks} [{-threads | -load} #threads | -read.threads #load]]...	
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).
	-load	Changes the load of a rank, by default all ranks have load 1.
	#load	Load for the current rank.
Examples	−rank 1 -threads 1	
	-rank 2 -threads 8	
	-all.ranks -threads 48	
	-rank 2 -load 4	

Table 8.20: 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	−name $fluid	
Description	Name of flow medium.	
Parameters	$fluid	Flow medium.
Examples	−name air	

Table 8.21: equation.of.state name option

Type		
Syntax	−type $eos.type.fluid	
Description	Choose the type of equation of state.	
Parameters	$eos.type.fluid	Type of equation of state.
Examples	−type eos.ideal.gas	

Table 8.22: equation.of.state type option

R		
Syntax	−R #	
Description	Set the specific gas constant. Default is 287.22 (air).	
Parameters	#	Specific gas constant.
Examples	−R 287.22	

Table 8.23: equation.of.state R option

Gamma		
Syntax	−gamma #	
Description	Set the heat capacity ratio (γ). Default is 1.4 (air).	
Parameters	#	Heat capacity ratio.
Examples	−gamma 1.4	

Table 8.24: equation.of.state gamma option

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

Table 8.25: equation.of.state Mu1 option

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

Table 8.26: equation.of.state Mu2 option

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

Table 8.27: equation.of.state Ka1 option

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

Table 8.28: equation.of.state Ka2 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.V	Reference velocity. Default is 1.
	-ref.length	Reference length. Default is 1.
	#	Reference value.
Examples	−ref.p 101325	
	-ref.length 0.5	

Table 8.29: equation.of.state ref option

8.9 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	−name $name	
Description	Name of the boundary as set in the Star-CD input files.	
Parameters	$name	Name of boundary.
Examples	−name WALL	

Table 8.30: BC name option

Type		
Syntax	−type $type	
Description	Type of boundary condition. See Chapter 5 and Table 8.32 for available types.	
Parameters	$type	Type of boundary condition.
Examples	−type noslip.wall	

Table 8.31: BC type option

Boundary Condition Types	
BC type	Description (see Chapter 5 for details)
impermeable.wall	Impermeable wall.
noslip.wall	No slip wall.
riemann	Riemann type outer boundary conditions.
turkel	Turkel type outer 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 8.32: Boundary condition types

Log	
Syntax	−log $log [log-options] ...
Description	Report in log file. See Table 8.34 for log options list.
Parameters	$log Log name.
Examples	−log coefficients cl cd

Table 8.33: Boundary conditions log option

BC Log Options	
Log type	Description
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.
cmy	Moment coefficient about Y coordinate direction.
cmz	Moment coefficient about Z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.

Table 8.34: BC log options

8.9.1 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	−wall.distance
Description	Explicitly set wall distance search for a specific boundary. Default: for all walls.
Parameters	N/A
Examples	−wall.distance

Table 8.35: BC wall.distance option

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

Table 8.36: BC no.wall.distance option

8.10 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	−type $type	
Description	Type of equation set to be solved.	
Parameters	$type	Type of equation set. See Table 8.38 for equation system types.
Examples	−type ideal.gas.mf.inviscid	

Table 8.37: 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.
turbulent.kw.tnt	Solve the $k-\omega$-TNT turbulence model equations.

Table 8.38: System types

Cell.gradient		
Syntax	−cell.gradient $method	
Description	Method to calculate the cell center gradient that is required for high order approximations.	
Parameters	$method	Method for cell gradient calculation. See Table 8.40 for a list of available cell gradient methods.
Examples	−cell.gradient green.gauss.node	

Table 8.39: 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 8.40: 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 8.42 for a list of available face gradient methods.
Examples	−face.gradient diamond	

Table 8.41: 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 8.42: Face gradient types

Limiter		
Syntax	−limiter $limiter	
Description	Type of limiter.	
Parameters	$limiter	Type of limiter. See Table 8.44 for limiter types.
Examples	−limiter venka.2d	

Table 8.43: 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 8.44: Limiter types

Venka.k		
Syntax	−venka.k #k	
Description	Change the Venkatakrishnan limiter coefficient. Default is 5.	
Parameters	#k	Venkatakrishnan limiter coefficient.
Examples	−venka.k 5	

Table 8.45: System venka.k option

Time.step		
Syntax	−time.step #initial #iterations {cfl.linear | cfl.exponential | cfl.tangential | dt.linear | dt.exponential | dt.tangential } #final	
Description	Sets the CFL number or time step (first order single time stepping).	
Parameters	#initial	Initial CFL number or time step.
	#iterations	Number of iteration for which the CFL number or time step grows from #initial to #final.
	cfl.linear	Linear growth of the CFL number.
	cfl.exponentia	Exponential growth of the CFL number.
	cfl.tangential	Hyperbolic tangent growth of the CFL number.
	dt.linear	Linear growth of the time step.
	dt.exponentia	Exponential growth of the time step.
	dt.tangential	Hyperbolic tangent growth of the time step.
	#final	Final CFL number or time step.
Examples	−time.step 5 50 cfl.linear 10	

Table 8.46: System CFL option

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

Table 8.47: System implicit.jacobi option

Implicit.save.diagonal	
Syntax	−implicit.save.diagonal
Description	Saves the diagonal for increased speed (higher memory requirement).
Parameters	N/A
Examples	−implicit.save.diagonal

Table 8.48: System implicit.save.diagonal option

Convection.noslip.diagonal		
Syntax	−convection.noslip.diagonal {normal | analytic | implicit.on.noslip}	
Description	Selects the way the diagonal convective jacobian is estimated on noslip boundaries.	
Parameters	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	−convection.noslip.diagonal analytic	

Table 8.49: System convection.noslip.diagonal option

Convection.impermeable.diagonal		
Syntax	−convection.impermeable.diagonal {normal | analytic | implicit.on.noslip}	
Description	Selects the way the diagonal convective jacobian is estimated on impermeable boundaries.	
Parameters	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	−convection.impermeable.diagonal analytic	

Table 8.50: System convection.impermeable.diagonal option

Convection.symmetry.diagonal		
Syntax	−convection.symmetry.diagonal {normal | analytic | implicit.on.noslip}	
Description	Selects the way the diagonal convective jacobian is estimated on symmetry boundaries.	
Parameters	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	−convection.symmetry.diagonal analytic	

Table 8.51: System convection.symmetry.diagonal option

Realizability.trb.w	
Syntax	−realizability.trb.w
Description	Adds realizability condition for omega (applies only to $k-\omega$ turbulence models).
Parameters	N/A
Examples	−realizability.trb.w

Table 8.52: 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 8.53: System source.mpk option

Damp	
Syntax	−damp #damp-factor
Description	Set damping to the convection Jacobian. Values that are less than unity increase stability. The recommended damping factor is 0.5-1.0. Default is 1.
Parameters	#damp-factor Damping factor.
Examples	−damp 0.75

Table 8.54: System damp option

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

Table 8.55: System relaxation option

iteration.convergence	
Syntax	−iteration.convergence #criterion
Description	Set the convergence criterion (log(Res/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	#criterion Convergence criterion in terms of (log(Res/Res0)).
Examples	−iteration.convergence -5

Table 8.56: System iteration.convergence option

Log	
Syntax	−log $log [log-options] ...
Description	Report in log file. See Table 8.58 for log options list.
Parameters	$log Log name.
Examples	−log coefficients cl cd

Table 8.57: System log option

System Log Options	
Log type	Description
cfl	System CFL.
dt	System Δt.
residual	System residual.
log.residual	System log of the residual.
max(mt)	Max of the turbulence viscosity (where applicable).

Table 8.58: System log options

Plot	
Syntax	−plot $plot [plot-options] ...
Description	Functions to include in FV-UNS file. See Table 8.60 for plot functions list.
Parameters	$plot Plot name.
Examples	−plot NACA0012 r p velocity residual

Table 8.59: System plot option

Plot Functions

Plot functions	Description
residual	Residual 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).
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).

Table 8.60: Plot functions

8.11 Solve Options

The solve input options section start with the −−solve directive, followed by a series of solve options.

Convection.flux		
Syntax	−convection.flux $method	
Description	Set the convection flux approximation method.	
Parameters	$method	Flux approximation method. See Table 8.62 for available convection flux types.
Examples	−convection.flux hllc.roe	

Table 8.61: Solve convection.flux option

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

Table 8.62: Convection flux types

Convection.jacobian	
Syntax	−convection.jacobian $jacobian
Description	Set the convection Jacobian. See Table 8.64 for available convection Jacobian types.
Parameters	$jacobian Jacobian.
Examples	−convection.jacobian hllc.roe

Table 8.63: Solve convection.jacobian option

Convection Jacobian Types	
Convection Jacobian type	Description
hllc.roe	HLLC Roe.
hllc.davis	HLLC Davis.
van.leer	van Leer.

Table 8.64: Convection Jacobian types

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

Table 8.65: Solve spatial.order option

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

Table 8.66: Solve sweeps option

Time.march		
Syntax	−time.march $method	
Description	Set the time marching method. See Table 8.68 for a list of available time marching methods.	
Parameters	$method	Time marching method.
Examples	−time.march implicit.pgs	

Table 8.67: 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.

Table 8.68: Time marching methods

Iterations		
Syntax	−iterations #	
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	#	Number of iterations.
Examples	−iterations 1000	

Table 8.69: Solve iterations option

Dual.time		
Syntax	−dual.time #steps #time-step	
Description	Set dual time stepping simulation.	
Parameters	#steps	Number of physical time steps.
	#time-step	Physical time step (Δt).
Examples	−dual.time 200 0.0001	

Table 8.70: Solve dual.time option

Save		
Syntax	−save #	
Description	Set the intermittency of output of restart files.	
Parameters	#	Number of steps between output.
Examples	−save 10	

Table 8.71: Solve save option

Save.path	
Syntax	−save.path $path
Description	Set the path for output of log and restart files.
Parameters	$path Path of log and restart files.
Examples	−save.path ./save/

Table 8.72: Solve save.path option

Save.sequential	
Syntax	−save.sequential
Description	Adds iteration signature to the saves.
Parameters	N/A
Examples	−save.sequential

Table 8.73: Solve save.sequential option

Load.path	
Syntax	−load.path $path
Description	Set the path for restart files.
Parameters	$path Path of restart files.
Examples	−load.path ./load/

Table 8.74: Solve load.path option

Load.sequence		
Syntax	−load.sequence #iteration	
Description	Load a specific iteration.	
Parameters	#iteration	Iteration for load.
Examples	−load.sequence 1000	

Table 8.75: Solve load.sequence option

EOS		
Syntax	−eos $eos	
Description	Set the equation of state.	
Parameters	$eos	Equation of state. Currently, only perfect gases are supported.
Examples	−eos air	

Table 8.76: Solve eos option

P		
Syntax	−p #	
Description	Set the free stream pressure.	
Parameters	#	Free stream pressure.
Examples	−p 101325	

Table 8.77: Solve p option

T		
Syntax	−T #	
Description	Set the free stream temperature.	
Parameters	#	Free stream temperature.
Examples	−p 288	

Table 8.78: Solve T option

Mach		
Syntax	−Mach #	
Description	Set the free stream Mach number.	
Parameters	#	Free stream Mach number.
Examples	−Mach 0.8	

Table 8.79: Solve Mach option

Velocity.magnitude		
Syntax	−velocity.magnitude #	
Description	Set the free stream velocity magnitude.	
Parameters	#	Free stream velocity magnitude.
Examples	−velocity.magnitude 200	

Table 8.80: Solve velocity magnitude option

U		
Syntax	−u #	
Description	Set the free stream X coordinate direction velocity component.	
Parameters	#	Free stream X coordinate direction velocity component.
Examples	−u 200	

Table 8.81: Solve u option

V		
Syntax	−v #	
Description	Set the free stream Y coordinate direction velocity component.	
Parameters	#	Free stream Y coordinate direction velocity component.
Examples	−v 0	

Table 8.82: Solve v option

W		
Syntax	−w #	
Description	Set the free stream Z coordinate direction velocity component.	
Parameters	#	Free stream Z coordinate direction velocity component.
Examples	−w 20	

Table 8.83: Solve w option

Alpha		
Syntax	−alpha #	
Description	Set the free stream angle of attack.	
Parameters	#	Free stream angle of attack in degrees.
Examples	−alpha 10	

Table 8.84: Solve alpha option

Beta		
Syntax	−beta #	
Description	Set the free stream side slip angle.	
Parameters	#	Free stream side slip angle in degrees.
Examples	−beta 0	

Table 8.85: 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 8.86: 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 8.87: Solve trb.mt option

Ref		
Syntax	−ref.$ # { # # }	
Description	Set the reference length, area, and reference point for force and moment coefficient calculations.	
Parameters	-reference.area	Reference area.
	-reference.len	Reference length.
	-reference.point	Reference point.
	#	Reference value.
Examples	−reference.len 0.325	
	-reference.point 1 0 0	

Table 8.88: Solve ref option

Reference.velocity		
Syntax	−reference.velocity #ref	
Description	Set the reference velocity. Default is the mean flow free stream velocity.	
Parameters	#ref	Reference velocity.
Examples	−reference.velocity 300	

Table 8.89: Solve reference.velocity option

Reference.longitudinal	
Syntax	−reference.longitudinal #ref
Description	Set the reference length for Reynolds number evaluation. Unused by the current version.
Parameters	#ref Reference length.
Examples	−reference.length 1.5

Table 8.90: Solve reference.length option

Log	
Syntax	−log $log [log-options] ...
Description	Report in log file. See Table 8.92 for log options list.
Parameters	$log Log name.
Examples	−log coefficients cl cd

Table 8.91: 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.
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.
cmy	Moment coefficient about Y coordinate direction.
cmz	Moment coefficient about Z coordinate direction.
cl	Lift coefficient.
cd	Drag coefficient.

Table 8.92: Solve log options

8.12 UNS Options

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

Name	
Syntax	−name $name
Description	Name of flow case.
Parameters	$name Flow case name.
Examples	−name naca0012_laminar

Table 8.93: UNS name option

Prefix	
Syntax	−prefix $prefix
Description	Set prefix of the flow case.
Parameters	$prefix Prefix of flow case.
Examples	−prefix ./naca0012/NACA0012-Str-Laminar-Rey500

Table 8.94: UNS prefix 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 8.95: UNS scale option

Min.parts		
Syntax	−min.parts #	
Description	Set minimum number of partitions.	
Parameters	#	Minimum number of partitions.
Examples	−min.parts 8	

Table 8.96: UNS min.parts option

Max.parts		
Syntax	−max.parts #	
Description	Set maximum number of partitions.	
Parameters	#	Maximum number of partitions.
Examples	−max.parts 8	

Table 8.97: UNS max.parts option

8.13 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.
time.step [$system] #	Change to the specified constant CFL or time step of all the systems or the specified $system.
plot	Dump the FVUNS file as requested by the user at the next available opportunity.
save	Save a restart

Table 8.98: Run-time options

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] P. L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.

[6] SF Davis. Simplified second-order godunov-type methods. *SIAM Journal on Scientific and Statistical Computing*, 9(3):445–473, 1988.

[7] Meng-Sing Liou and Christopher J. Steffen Jr. A new flux splitting scheme. *Journal of Computational Physics*, 107(1):23–39, 1993.

[8] Meng-Sing Liou. A sequel to AUSM: $AUSM^+ - up$. *Journal of Computational Physics*, 129:364–382, 1996.

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