POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

MSc of Mathematical Engineering

Tesi di Laurea Magistrale

Development of a reactive numerical solver based on the SU2 CFD code

Relatore:
Prof. Paolo Francesco BARBANTE

Giuseppe ORLANDO
Matr. 878776

Academic Year 2018-2019

Contents

List of Symbols

δ_{ij} = Kronecker's delta
p = Mixture pressure
ρ = Mixture density
ρ_i = Density of species i
Y_i = Mass fraction of species i
X_i = Mole fraction of species i
T = Temperature
$\boldsymbol{u}$ = Velocity
u_x = Component of the velocity along x direction
u_y = Component of the velocity along y direction
u_z = Component of the velocity along z direction
$\frac{\partial \cdot}{\partial(\rho \boldsymbol{u})} = \begin{pmatrix} \frac{\partial \cdot}{\partial(\rho u_x)} \\ \frac{\partial \cdot}{\partial(\rho u_y)} \\ \frac{\partial \cdot}{\partial(\rho u_z)} \end{pmatrix}$
E = Total energy per unit of mass
e = Internal energy per unit of mass
H = Mixture total enthalpy
h = Mixture static enthalpy
h_i = Static enthalpy of species i
C_p = Mixture specific heat at constant pressure
C_v = Mixture specific heat at constant volume
C_{p_i} = Specific heat at constant pressure of species i
μ = Mixture laminar viscosity
k = Mixture thermal conductivity
$k_{f,r}$ = Forward rate of reaction r
$k_{b,r}$ = Backward rate of reaction r
K_r = Equilibrium constant of reaction r
A_r = Pre-exponential factor or reaction r
β_r = Temperature exponent of reaction r
E_r = Activation energy of reaction r
T_r = Activation temperature of reaction r
$\dot{\omega}_i$ = Mass production term of species i
$[\cdot]$ = Concentration of species $\cdot$

List of Figures

List of Tables

Ringraziamenti

Ringrazio il professor Barbante per avermi seguito con grande pazienza nella stesura di questa tesi e per avermi incoraggiato e supportato con utili consigli e suggerimenti, che hanno migliorato questo lavoro.

Sommario

La tesi si occupa della formulazione e dell'implementazione delle equazioni che modellano fluidi reattivi.
Particolare attenzione è stata posta alla modellazione dei flussi di diffusione molecolare che sono coloro che "alimentano la chimica" usando risultati rigorosi provenienti dalla teoria cinetica dei gas.
Per questo motivo le celebri equazioni di **Stefan-Maxwell** sono state utilizzate per calcolare i già menzionati flussi di diffusione. Il problema con questo tipo di modello è che sorgono singolarità a causa delle proprietà di conservazione dei flussi, specialmente nel caso di frazioni di massa nulle o evanescenti; pertanto per superare questa difficoltà vengono proposte delle modifiche ad hoc delle equazioni di Stefan-Maxwell.

L'implementazione è stata eseguita sulla suite **SU2**: si tratta di un codice per simulazioni di Fluidodinamica Computazionale fornito dall'Università di Stanford. In questo lavoro le sue potenzialità sono state estese per poter studiare fluidi reattivi con lo sviluppo inoltre di una libreria per il calcolo di proprietà fisiche e chimiche.

Pertanto l'elaborato è stato strutturato come segue: nel primo capitolo introduciamo i principi fisici generali tipici dei fluidi reattivi; poi nel secondo formuliamo il modello matematico che governa il fenomeno.
Successivamente presentiamo il metodo numerico adottato per discretizzare le equazioni: verrà usato uno schema ai volumi finiti di tipo vertex-centered con l'utilizzo dello schema di AUSM per i flussi convettivi.
Nel quarto capitolo è riportata una descrizione generale riguardo SU2 e le sue funzionalità in modo da introdurre brevemente le aggiunte richieste per trattare fluidi reattivi ed anche la libreria implementata.
Infine alcuni test vengono eseguiti in modo da validare il codice: sono riportati in particolare i risultati relativi a simulazioni dei processi di combustione che avvengono all'interno dei motori aerospaziali ed a un caso di rientro.

Abstract

This thesis explores the formulation and implementation of the equations that model chemically reacting flows.
Specific attention has been paid to the modelling of molecular diffusion fluxes, which are the one that "feed the chemistry" using rigorous results of the kinetic theory of gases. For this purpose the well-known complete **Stefan-Maxwell** equations have been used in order to compute the diffusion fluxes. The problem with this kind of model is that singularities arise in the equation due to the conservation property of fluxes, especially in case of zero or vanishing mass fractions: in order to overcome this issue ad hoc modifications of Stefan-Maxwell equations are presented.

The implementation has been performed on the **SU2** suite: it is a general purpose Computational Fluid Dynamics code provided by the Stanford University. In this work its capabilities have been extended in order to study multispecies reacting flows with the development of a library to compute physical and chemical properties.

Therefore the thesis has been structured in this way: in the first chapter we introduce the general physical principles that are typical of reacting flows; then in second chapter we formulate the mathematical model that governs this kind of phenomenon.
Subsequently we present the numerical method employed to discretize the governing equations: it will be used a vertex-centered finite volume scheme with the employment of AUSM scheme for convective fluxes.
Then a general description about SU2 and its functionalities is performed in order to briefly illustrate the additions required for reactive flows and also the implemented library.
Eventually some test cases are performed in order to validate the code, in particular simulations of the combustion processes that take place inside aerospace engines and a re-entry case.

Chapter 1

Introduction

1.1 Physical Processes

Two main physical processes are involved in a reacting flow: the fluid dynamics and the chemical reactions.
The fluid dynamics process is the balance between the temporal evolution and the spatial convection of the flow properties due to conservation of mass, momentum, and energy, while chemical reactions determine the generation/destruction of chemical species under the constraint of mass conservation.
Each of the above processes could be either evolving or in equilibrium. For the evolving condition, each above process has its own space and time scales, and they are very different from that of other processes. Such differences in space and time scales, on one hand, could allow simplification in the theoretical model. On the other hand, they could be the source of numerical difficulties. In this work, we assume that the space and time scales of fluid dynamics and chemical reactions are much larger than that of thermodynamics. Thus, the thermodynamic process is always considered to be in equilibrium. From the viewpoint of thermodynamics, the chemical composition of the reactive gas mixture is locally frozen, and the gas mixture is locally motionless. We refer to this condition as thermal equilibrium. In the following subsections, we provide further discussions about the thermal equilibrium assumption and the interaction between fluid dynamics and chemical reactions.

1.2 Local Thermal Equilibrium

Due to the assumption of thermal equilibrium, the concept of state variables and the equation of state in the classical thermodynamics can be used to provide the relationship between thermodynamic variables. Because of flow motion and the associated pressure and temperature distribution throughout the space-time domain, the flow field as a whole is not in thermal equilibrium. The notion of the classical thermodynamics must be supplemented by additional ideas: the whole flow system is subdivided into a large number of subsystems that are small as compared to the whole system but still

of macroscopic size relative to the molecular structure of the fluid. By assuming that each subsystems is in thermal equilibrium internally but not in equilibrium with its neighbours, we then apply equilibrium thermodynamics to each subsystem. As a result, we can build up, by integrating the flow equations, the space-time evolution of the entire non-thermally-equilibrium system. Physically, the above notion implies that numerous molecular collisions occur locally during a typical time increment of a CFD calculation such that various energy modes, i.e., translation, rotation, vibration, of individual molecule and the thermal energy between different molecules (including different species) are statistically in equilibrium. Thus, there is only one temperature of the gas mixture at each space-time location. We remark that this assumption may not be true for hypersonic (rarefied) flows, in which the time scales associated with molecular collisions and the mean free path may be comparable to the flow residence time and the associated length scales. Thus, there are not enough collisions between molecules within each time step for the local system to reach a thermally equilibrium state.

1.3 Flow/Reaction Interactions

The interactions between chemical reactions and fluid dynamics is best described by the Damköhler number, which is the ratio of the characteristic time of fluid mechanics to that of chemical reactions. As the Damköhler number approaches infinity, the chemical reactions are much faster compared to fluid dynamics. Thus, the chemical composition can be treated as a state variable governed by the chemical equilibrium theory, i.e., free energy minimization. In this case, the above thermal equilibrium needs to be modified in order to include the effect of free energy minimization. Thus the fluid dynamics becomes the only evolving process. The discussion of this condition, however, is out of the scope of the present work.

If Damköhler number is close to zero, the chemical reactions are slow as compared to fluid flows, and a non-reactive fluid can be assumed. Only when the Damköhler number is of the order of unity, does one anticipate the greatest interaction between chemical reactions and fluid dynamics. In this case, both fluid dynamics and chemical reactions should be treated as evolving processes, and one must use the finite-rate kinetics to model chemical reactions. We shall focus on this condition in the present work.

However, in the finite rate chemistry with multiple reaction steps, a wide range of time scales for chemical reactions exists. Thus, the Damköhler numbers associated with individual reaction steps span a wide range of values. In this case, a global Damköhler number could be assigned based on the bottle neck reaction step, which can be deduced by a sensitivity study of the chemical kinetics. Nevertheless, some confusion remains in defining a single Damköhler number when using multiple reaction steps.

In this work, and as always in other CFD works for combustion, the space-time evolution of fluid dynamics is our main concern, and the space and time scales of fluid dynamics will be used as the pacing parameters in the calculations. In addition, the time increment and spatial mesh employed will also be constrained by the CFL number condition for numerical stability. Because of the wide range of the time and space scales associated

with the multiple reaction steps, such a choice of the time step and the mesh size based on the CFL number constraint could grossly under-resolve some of the reaction steps. This will cause severe stiffness problem in numerical calculations.
The common practice is to resort to special treatments for the stiff source terms in the species equations without fully resolving the space and time scales of the source terms.

Chapter 2

Mathematical Model

2.1 Conservative Governing Equations

We consider first the unsteady, inviscid and chemically reacting flow equations in three spatial dimensions:

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot \mathbf{E} = \mathbf{S} \tag{2.1}$$

where

$$\boldsymbol{Q} = \begin{pmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho E \\ \rho_1 \\ \rho_2 \\ ... \\ \rho_{N_s} \end{pmatrix}, \boldsymbol{E} = \begin{pmatrix} \rho \boldsymbol{u} \\ \rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} \\ (\rho E + p)\boldsymbol{u} \\ \rho_1 \boldsymbol{u} \\ \rho_2 \boldsymbol{u} \\ ... \\ \rho_{N_s} \boldsymbol{u} \end{pmatrix}, \boldsymbol{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ ... \\ \dot{\omega}_{N_s} \end{pmatrix}$$

Here $\otimes$ denotes the tensor product and $\nabla\cdot$ denotes the divergence operator.

This set of equations is known as **Euler** equations: the first three are the continuity, momentum and energy equations respectively, and the rest are N_s species continuity equations, describing the mass conservation of each gas species.
In (2.1), ρ is the density of the gas mixture, while ρ_{i} is the density for species $i, \boldsymbol{u} = (u_x, u_y, u_z)$ is the velocity of the gas mixture and E is the total energy of the gas mixture per unit mass, and is defined as

$$E = e + \frac{\|\mathbf{u}\|^2}{2} \tag{2.2}$$

where $\|\cdot\|$ denotes the Euclidean norm and e is the internal energy of the gas mixture per unit mass: it is computed based on a mass-weighted average of the internal energy per unit mass of each species e_i, i.e.,

$$e = \sum_{i=1}^{N_s} Y_i e_i. \tag{2.3}$$

Note that

$$Y_i = \frac{\rho_i}{\rho}$$

is the mass fraction of species i in the gas mixture.
As it will be shown later, the definitions of internal energy e and total energy E include the heat of formation of chemical species. Therefore, no source term exists in the energy equation. Moreover, mass is conserved through chemical reactions and the summation of all source terms is zero, i.e.

$$\sum_{i=1}^{N_s} \dot{\omega_i} = 0$$

as we will see later on.
Eventually we assume that individual species behave as thermally perfect gases, i.e. they satisfy the relation:

$$p_i = \rho_i R_i T \tag{2.4}$$

where p_i is the partial pressure, R_i is the gas constant of species i and T is the temperature. The gas constant R_i is computed based on

$$R_i = \frac{R_u}{M_i}$$

where $R_u = 8.31 \quad \mathrm{J\,mol^{-1}\,K^{-1}}$ is the universal gas constant and M_i is the molar mass of species i.
As Dalton's law prescribes that the pressure p of a mixture is equal to the sum of partial pressure, we find:

$$p = \sum_{i^1}^{N_s} p_i = \sum_{i=1}^{N_s} \rho_i R_i T = \sum_{i=1}^{N_s} \rho Y_i R_i T = \rho R T \tag{2.5}$$

where $R = \sum_{i=1}^{N_S} Y_i R_i$ is the gas constant of the mixture.

In case of a viscous reactive flow the governing equations that express the conservation of mass, momentum and energy become respectively:

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

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I}) - \nabla \cdot \boldsymbol{\tau} = 0 \tag{2.7}$$

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot ((\rho E + p)\boldsymbol{u}) - \nabla \cdot (\boldsymbol{\tau}\boldsymbol{u} - \boldsymbol{q}) = 0 \tag{2.8}$$

Here $\boldsymbol{\tau}$ is the stress tensor whose definition is

$$\boldsymbol{\tau} = \mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right) - \left(\eta - \frac{2}{3}\mu \right) (\nabla \cdot \boldsymbol{u})\, \boldsymbol{I}$$

where μ is the laminar viscosity and η is the volume viscosity, and $\boldsymbol{q}$ is the heat flux due to conduction and diffusion whose expression will be specified later on.
In this work we consider negligible the contribution due to η, applying the so-called Stokes hypothesis which has become a common practice in the analysis of the motion of compressible fluids.

The continuity equation for a single species i is modified as follows:

$$\frac{\partial \rho_i}{\partial t} + \nabla \cdot (\rho_i \boldsymbol{u}) + \nabla \cdot \boldsymbol{J}_i = \dot{\omega}_i \tag{2.9}$$

In equation (2.9) the term $\boldsymbol{J}_i$ represents the diffusion flux of species i whose contribution must be taken into account in the expression of $\boldsymbol{q}$:

$$\boldsymbol{q} = -k\nabla T + \sum_{i=1}^{N_S} h_i \boldsymbol{J}_i$$

where k represents the thermal conductivity and h_i is the static enthalpy of species i (including heat of formation) defined as:

$$h_i = e_i + \frac{p_i}{\rho_i} \tag{2.10}$$

The static enthalpy can be related to the specific heat at constant pressure C_{p_i}thanks to first law of thermodynamics; indeed if we define

$$C_{p_i} = \left.\frac{\partial q_i}{\partial T}\right|_{p_i}$$

we find that

$$dq_i = T ds_i = de_i + p_i dv_i = dh_i - v_i dp_i$$

Hence it follows immediately that

$$C_{p_i} = \left.\frac{\partial h_i}{\partial T}\right|_{p_i} \tag{2.11}$$

Herein q_i is the reversible heat transfer per unit of mass, s_i is the entropy of species i and v_i is the volume occupied by species i.
We immediately note that h_i is a function of T only and can be proper expressed as integration of (2.11):

$$h_i = \int_{T_{ref}}^{T} C_{p_i} dT + h_{f_i} \tag{2.12}$$

where T_{ref} is a reference temperature and h_{f_i} is the assigned value of enthalpy at $T = T_{ref}$ (**formation enthalpy**).
Moreover the following conservation constraint must hold

$$\sum_{i=1}^{N_S} \boldsymbol{J}_i = 0$$

in order to ensure the total mass conservation as expressed by (2.6).

In case the dilute approximation (also known as Fick's law) holds we can say through Ramshaw self-consistent modification of fluxes that

$$\boldsymbol{J}_i = -\rho \frac{M_i}{M} D_{i,m} \nabla X_i + Y_i \sum_{j=1}^{N_S} \rho D_{i,m} \frac{M_j}{M} \nabla X_j \tag{2.13}$$

where X_i is the molar fraction of species i and $D_{j,m}$ is the mass diffusion coefficient for species i in the mixture.
In other cases the dilute approximation may not be acceptable and full multicomponent diffusion is required; in such cases, Stefan-Maxwell equations must be solved:

$$\boldsymbol{d}_i + \theta_i \frac{\nabla T}{T} = \sum_{j=1}^{N_S} \frac{X_i X_j}{D_{ij}} \left(\boldsymbol{V_j} - \boldsymbol{V_i} \right) \tag{2.14}$$

where $\boldsymbol{V_i}$ and $\boldsymbol{V_j}$ represent the diffusion velocity of species i and j respectively, θ_i is the thermal diffusion ratio of species i, while $\boldsymbol{d}_i$ can be expressed as

$$\boldsymbol{d}_i = \nabla X_i + (X_i - Y_i) \nabla \ln p - \frac{Y_i}{p} \left(\rho \boldsymbol{f_i} - \rho \sum_{k=1}^{N_S} Y_k \boldsymbol{f_k} \right) \tag{2.15}$$

$\boldsymbol{f}$ are body forces terms.
In this work no body forces are considered and we neglect the contribution due to the pressure so that:

$$\boldsymbol{d}_i = \nabla X_i \tag{2.16}$$

and moreover we neglect the contribution due to thermal diffusion in the Stefan-Maxwell equations.
Anyway it is worth to underline that the considerations about Stefan-Maxwell equations that we will discuss later on hold even in case all terms of (2.14) are considered.
Summing up we can write the governing equations as

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot \mathbf{F} - \nabla \cdot \mathbf{G} = \mathbf{S} \tag{2.17}$$

where

$$\boldsymbol{Q} = \begin{pmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho E \\ \rho_1 \\ \rho_2 \\ \dots \\ \rho_{N_s} \end{pmatrix}, \boldsymbol{F} = \begin{pmatrix} \rho \boldsymbol{u} \\ \rho \boldsymbol{u} \otimes \boldsymbol{u} + p \boldsymbol{I} \\ (\rho E + p) \boldsymbol{u} \\ \rho_1 \boldsymbol{u} \\ \rho_2 \boldsymbol{u} \\ \dots \\ \rho_{N_S} \boldsymbol{u} \end{pmatrix}, \boldsymbol{G} = \begin{pmatrix} 0 \\ \boldsymbol{\tau} \\ \boldsymbol{\tau u} - \boldsymbol{q} \\ -\boldsymbol{J}_1 \\ -\boldsymbol{J}_2 \\ \dots \\ -\boldsymbol{J}_{N_S} \end{pmatrix}, \boldsymbol{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dots \\ \dot{\omega}_{N_s} \end{pmatrix}$$

and the set of equations (2.17) is known as **Navier-Stokes** equations.

It is not common to consider as independent variables both the mixture density ρ and the partial densities ρ_i of all N_S species since this choice seems to be redundant: the reason is that eventual mistakes in the chemical composition due to the approximate numerical algorithm are not modified by the convective terms and also the diffusion part, as we will see later on, is not able to dissipate these errors because of the intrinsic structure of Stefan-Maxwell equations.

2.2 Analysis of Stefan-Maxwell Equations

Let us try now to develop the expression of Stefan-Maxwell equations in order to obtain a linear system in terms of $\boldsymbol{J}_i$.
First of all let us rewrite (2.14) in a more convenient way; since $\boldsymbol{J}_i = \rho_i \boldsymbol{V_i}$ we get:

$$\begin{aligned}\boldsymbol{d}_i &= \sum_{j=1}^{N_S} \frac{X_i X_j}{D_{ij}} \left(\frac{\boldsymbol{J}_j}{\rho_j} - \frac{\boldsymbol{J}_i}{\rho_i} \right) = \sum_{j=1}^{N_S} \frac{X_i X_j}{D_{ij}} \left(\frac{\boldsymbol{J}_j}{\rho Y_j} - \frac{\boldsymbol{J}_i}{\rho Y_i} \right) = \sum_{j=1}^{N_S} \frac{X_i X_j}{D_{ij}} \left(\frac{M\boldsymbol{J}_j}{\rho M_j X_j} - \frac{M\boldsymbol{J}_i}{\rho M_i X_i} \right) \\ &= \frac{M}{\rho} \sum_{j=1}^{N_S} \frac{X_i X_j}{D_{ij}} \left(\frac{\boldsymbol{J}_j M_i X_i - \boldsymbol{J}_i M_j X_j}{X_i X_j M_i M_j} \right) = \frac{M}{\rho} \sum_{j=1}^{N_S} \frac{\boldsymbol{J}_j M_i X_i - \boldsymbol{J}_i M_j X_j}{D_{ij} M_i M_j} \end{aligned} \tag{2.18}$$

From the conservation constraint we can eliminate $\boldsymbol{J}_{N_S}$ from the set of Stefan-Maxwell equations and reduce to $N_S - 1$ relations, namely

$$\sum_{j=1}^{N_S} \boldsymbol{J}_j = 0 \implies \boldsymbol{J}_{N_S} = -\sum_{j=1}^{N_S-1} \boldsymbol{J}_j = -\boldsymbol{J}_i - \sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \boldsymbol{J}_j$$

We can exploit the manipulation provided in (2.18) to say that

$$c\boldsymbol{d}_i = -\frac{\boldsymbol{J}_i}{M_i} \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}} + X_i \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{\boldsymbol{J}_j}{M_j D_{ij}}$$

where $\frac{1}{M} = \sum_{i=1}^{N_S} \frac{Y_i}{M_i}$ is the inverse of the mixture molar mass and $c = \frac{\rho}{M}$ is the mixture concentration.
Now we can finally isolate and then eliminate the N_S^{th} diffusive flux

$$\begin{aligned} c\boldsymbol{d}_i &= -\frac{\boldsymbol{J}_i}{M_i} \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}} + X_i \sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \frac{\boldsymbol{J}_j}{M_j D_{ij}} + X_i \frac{\boldsymbol{J_{N_S}}}{M_{N_S} D_{iN_S}} \\ &= -\frac{\boldsymbol{J}_i}{M_i} \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}} + X_i \sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \frac{\boldsymbol{J}_j}{M_j D_{ij}} - \frac{X_i}{M_{N_S} D_{iN_S}} \left(\boldsymbol{J}_i + \sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \boldsymbol{J}_j \right) \\ &= -\frac{\boldsymbol{J}_i}{M_i} \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}} - \frac{X_i}{M_{N_S} D_{iN_S}} \boldsymbol{J}_i + X_i \sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \frac{\boldsymbol{J}_j}{M_j D_{ij}} - \frac{X_i}{M_{N_S} D_{iN_S}} \sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \boldsymbol{J}_j \\ &= -\boldsymbol{J}_i \left[\frac{X_i}{M_{N_S} D_{iN_S}} + \frac{1}{M_i} \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}} \right] + X_i \left[\sum_{\substack{j=1 \\ j\neq i}}^{N_S-1} \boldsymbol{J}_j \left(\frac{1}{M_j D_{ij}} - \frac{1}{M_{N_S} D_{iN_S}} \right) \right] \end{aligned} \tag{2.19}$$

which can be expressed as a linear system of the form

$$\boldsymbol{BJ} = -c\boldsymbol{d}_i \tag{2.20}$$

where

$$B_{ij} = -X_i \left(\frac{1}{M_j D_{ij}} - \frac{1}{M_{N_S} D_{iN_s}} \right) \qquad i = 1...N_S$$

$$B_{ij} = \frac{X_i}{M_{N_S} D_{iN_S}} + \frac{1}{M_i} \sum_{\substack{j=1 \\ j \neq i}}^{N_S - 1} \frac{X_j}{D_{ij}} \qquad i, j = 1...N_S, \quad i \neq j$$

This approach has a significant drawback: you have to choose which of the $N_S - 1$ species you solve and so there is the presence of an asymmetry since one species is treated differently from the other and moreover some problems can arise from a numerical point of view if the selected species is not present in excess.

An alternative is represented by the iterative method proposed by Sutton in [5].
First of all let us rewrite the Stefan-Maxwell equations for $\boldsymbol{J}_i$ in a more convenient form:

$$\boldsymbol{d}_i = \frac{M}{\rho} \sum_{j=1}^{N_S} \left(\frac{X_i \boldsymbol{J}_j}{M_j D_{ij}} - \frac{X_j \boldsymbol{J}_i}{M_i D_{ij}} \right) \tag{2.21}$$

and let us express now these relations in terms of $\boldsymbol{d}_i$:

$$\boldsymbol{J}_i = -\rho \frac{M_i}{M} \sum_{j=1}^{N_S} \frac{X_j}{D_{ij}} \boldsymbol{d}_i + Y_i M \sum_{j=1}^{N_S} \frac{X_j}{D_{ij}} \sum_{j=1}^{N_S} \frac{\boldsymbol{J}_j}{M_j D_{ij}} \tag{2.22}$$

The closure equation that expresses the conservation is imposed by the iterative scheme

$$\boldsymbol{J}_i^{N+1} = \boldsymbol{J}_i^N - Y_i \sum_{j=1}^{N_S} \boldsymbol{J}_j^N$$

where the term $\boldsymbol{J}_i^N$ is computed using (2.22) for each species and the entire set is corrected for iteration $N + 1$ using the aforementioned conservation constraint.

Another approach has been proposed by Giovangigli in [24]: the analysis carried out takes into account some criticality of the Stefan-Maxwell equations once they are considered as a set of N_S independent relations.
Let us recall the Stefan-Maxwell equations as reported in (2.14):

$$\boldsymbol{d}_i = \sum_{j=1}^{N_S} \frac{X_i X_j}{D_{ij}} (\boldsymbol{V_j} - \boldsymbol{V_i}) = X_i \sum_{\substack{j=1 \\ j \neq i}}^{N_S} \frac{X_j \boldsymbol{V_j}}{D_{ij}} - X_i \boldsymbol{V_i} \sum_{\substack{j=1 \\ j \neq i}}^{N_S} \frac{X_j}{D_{ij}}$$

which can be expressed as a linear system of the form

$$\boldsymbol{F}\boldsymbol{V} = -\boldsymbol{d}_i \tag{2.23}$$

where

$$F_{ii} = X_i \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}} \qquad i = 1, ..., N_S$$

$$F_{ij} = -\frac{X_i X_j}{D_{ij}} \qquad i, j = 1, ..., N_S, \quad i \neq j$$

It is immediate to verify that $\sum_{j=1}^{N_S} F_{ij} = 0$ and therefore the matrix $\boldsymbol{F}$ is singular and hence not invertible.

The Stefan-Maxwell equations are equivalent [25] to the following relation:

$$\boldsymbol{V}_i = -\sum_{j=1}^{N_S} \mathcal{D}_{ij}\boldsymbol{d}_j \tag{2.24}$$

where $\boldsymbol{\mathcal{D}} = (\mathcal{D}_{ij})$ is the symmetric multicomponent diffusion matrix.
The elements $\mathcal{D}_{ij}$ are defined with the constraint $\sum_{j=1}^{N_S} Y_i\mathcal{D}_{ij} = 0$ as reported by [25].
Since $\mathcal{D}_{ij} = \mathcal{D}_{ji}, i, j = 1, ..., N_S$ the following relation holds:

$$\sum_{i=1}^{N_S} Y_i\mathcal{D}_{ij} = 0 \tag{2.25}$$

A direct consequence of (2.25) is that the relation

$$\sum_{i=1}^{N_S} Y_i\boldsymbol{V}_i = \boldsymbol{0} \implies \sum_{i=1}^{N_S} \boldsymbol{J}_i = \boldsymbol{0} \tag{2.26}$$

is satisfied independently on the driving forces $\boldsymbol{d}_i$. Therefore errors in the computation of the chemical composition (i.e. the mass fractions do not sum up to 1) cannot be dissipated by the diffusion fluxes.

The idea to overcome this issue is to modify the Stefan-Maxwell equations, in such a way that the constraints $\sum_{i=1}^{N_S} Y_i = 1$ and $\sum_{i=1}^{N_S} \boldsymbol{J}_i = \boldsymbol{0}$ are not imposed *a priori* but will be satisfied in the whole domain under the imposition of suitable boundary and initial conditions.

First of all since the continuity equations are solved for the mass fractions of the species, the driving forces of the Stefan-Maxwell equations must be expressed in terms of mass fractions as well. The original relation between the mole and mass fractions is given by:

$$X_i = \frac{M}{M_i} Y_i$$

so that

$$\nabla X_i = \frac{M}{M_i} \nabla Y_i + \frac{Y_i}{M_i} \nabla M$$

Since

$$\frac{1}{M} = \sum_{j=1}^{N_S} \frac{Y_j}{M_j}$$

then

$$\nabla M = -M^2 \sum_{j=1}^{N_S} \frac{\nabla Y_j}{M_j}$$

and so

$$\nabla X_i = \frac{M}{M_i} \nabla Y_i - M^2 \frac{Y_i}{M_i} \sum_{j=1}^{N_S} \frac{\nabla Y_j}{M_j} = \frac{M}{M_i} \nabla Y_i - M X_i \sum_{j=1}^{N_S} \frac{\nabla Y_j}{M_j} = \sum_{j=1}^{N_S} M_{ij} \nabla Y_j$$

where

$$M_{ii} = \frac{M}{M_i} (1 - X_i) \qquad i = 1...N_S$$
$$M_{ij} = -\frac{M X_i}{M_j} \qquad i, j = 1...N_S, \quad i \neq j$$

Note that $\sum_{i=1}^{N_S} M_{ij} = 0$ and so also the matrix $\boldsymbol{M} = M_{ij}$ is singular.
From another prospective if we couple the definition of mixture molar mass and the relation between mole and mass fraction, we find that $\sum_{i=1}^{N_S} X_i = 1$ independently from the values of mass fractions, i.e it is imposed a priori. Indeed:

$$X_i = \frac{M}{M_i} Y_i \implies \sum_{i=1} \frac{M}{M_i} Y_i = \frac{M}{M} = 1.$$

In order to eliminate the singularity of the aforementioned matrix we adopt the following relation between mass and mole fractions as proposed by [26]:

$$X_i = \sigma \frac{M}{M_i} Y_i$$

where $\sigma = \sum_{i=1}^{N_S} Y_i$.
Note that $\sum_{i=1}^{N_S} X_i = \sum_{i=1}^{N_S} Y_i$ and the gradient is now given by:

$$\nabla X_i = \frac{\sigma M}{M_i}\nabla Y_i - M X_i \sum_{j=1}^{N_S} \frac{\nabla Y_j}{M_j} + \frac{M Y_i}{M_i}\nabla\sigma = \widetilde{M_{ij}}\nabla Y_i$$

where

$$\widetilde{M_{ii}} = \frac{M}{M_i}\left(Y_i - X_i + \sigma\right) \qquad i = 1...N_S$$
$$\widetilde{M_{ij}} = M\left(\frac{Y_i}{M_i} - \frac{X_i}{M_j}\right) \qquad i,j = 1...N_S, \quad i \neq j$$

Note that $\sum_{i=1}^{N_S} \widetilde{M_{ij}} = 1$: the matrix $\widetilde{\boldsymbol{M}} = \widetilde{M_{ij}}$ is not singular and $\sum_{i=1}^{N_S} \nabla X_i = \sum_{i=1}^{N_S} \nabla Y_i$.

As shown by [24] singularities in the Stefan-Maxwell equations will appear for flux boundary conditions and homogeneous Neumann conditions as well. The singular behaviour is due to the lack of a diffusion term for the "*species*" $\sum_{i=1}^{N_S} Y_i$: adding an artificial diffusion would suppress the singularity and this is achieved by adding $\alpha Y_i (\alpha > 0)$ times the mass flux constraint to each Stefan-Maxwell equation that becomes:

$$\sum_{j=1}^{N_S} F_{ij}\boldsymbol{V_J} + \alpha Y_i \sum_{j=1}^{N_S} Y_j \boldsymbol{V_J} = -\nabla X_i \tag{2.27}$$

The modified Stefan-Maxwell equations can be expressed as:

$$\widetilde{\boldsymbol{F}}\boldsymbol{V} = -\boldsymbol{d}_i$$

where $\widetilde{\boldsymbol{F}} = \boldsymbol{F} + \alpha \boldsymbol{Y} \otimes \boldsymbol{Y}$.
As shown by Giovangigli the matrix $\widetilde{\boldsymbol{F}}$ is symmetric and positive definite and therefore it is invertible. If we introduce the matrix $\boldsymbol{G} = \widetilde{\boldsymbol{F}}^{-1}$ we find

$$\boldsymbol{V} = -\boldsymbol{G}\boldsymbol{d}_i$$

Switching from velocities to mass fluxes $\boldsymbol{J} = J_i$ since $J_i = \rho Y_i V_i$ we find

$$\boldsymbol{J} = -\boldsymbol{H}\boldsymbol{d}_i$$

where $\boldsymbol{H} = \boldsymbol{R}\boldsymbol{G}$ where $\boldsymbol{R} = \mathrm{diag}(\rho Y_1, ...\rho Y_{N_S})$.
For positive mass fractions the matrices $\boldsymbol{R}$ and $\boldsymbol{G}$ are non-singular and in this case $\boldsymbol{H}$ is not singular as well; however difficulties arise in the case of zero or vanishing mass

fractions: in this situation the matrices $\boldsymbol{R}$ and $\widetilde{\boldsymbol{F}}$ are singular or ill-conditioned: indeed since $\widetilde{F_{ij}} = F_{ij} + \alpha Y_i Y_j$ we find that

$$\sum_{j=1}^{N_S} \widetilde{F_{ij}} = \sum_{j=1}^{N_S} F_{ij} + \sum_{j=1}^{N_S} \alpha Y_i Y_j = \alpha Y_i \sigma$$

which is equal or tends to 0 in presence of at least one zero or vanishing mass fraction.

Therefore we need a rescaled version of Stefan-Maxwell equations in terms of diffusion fluxes. We are led to introduce the matrix $\boldsymbol{\Gamma}$ such that:

$$\Gamma_{ii} = \sigma \frac{M}{\rho M_i} \sum_{\substack{j=1 \\ j \neq i}}^{N_S} \frac{X_j}{D_{ij}}, \qquad i = 1...N_S \tag{2.28}$$

$$\Gamma_{ij} = -\sigma \frac{M}{\rho M_j} \frac{X_i}{D_{ij}} \qquad i, j = 1...N_S, \quad i \neq j \tag{2.29}$$

so that:

$$\boldsymbol{\Gamma J} = -\boldsymbol{d}_i$$

It is important to notice that the mixture molar mass M must be computed using the following relation:

$$\frac{1}{M} = \sum_i Y_i M_i$$

and the other classical relation

$$M = \sum_i X_i M_i$$

does not hold anymore. Indeed if the previous formula were still valid we would obtain:

$$X_i M_i = \sigma M Y_i \implies M = \sigma M \sigma \implies \sigma^2 = 1 \implies \sigma = 1$$

which in general is not true.
Another possible approach would be considering

$$M = \sum_i X_i M_i$$

and define then

$$Y_i = \tilde{\sigma} \frac{M_i}{M} X_i$$

with $\tilde{\sigma} = \sum_{i=1}^{N_S} X_i$.
It would still hold that $\sum_{i=1}^{N_S} X_i = \sum_{i=1}^{N_S} Y_i$, but the entries of matrix $\boldsymbol{\Gamma}$ should be modified

as follows:

$$\widehat{\Gamma_{ii}} = \frac{M}{\rho M_i \tilde{\sigma}} \sum_{\substack{j=1 \\ j\neq i}}^{N_S} \frac{X_j}{D_{ij}}, \qquad i = 1...N_S \tag{2.30}$$

$$\widehat{\Gamma_{ij}} = -\frac{M}{\rho M_j \tilde{\sigma}} \frac{X_i}{D_{ij}} \qquad i,j = 1...N_S, \quad i \neq j \tag{2.31}$$

Coming back to the first formulation, we notice that since $\sum_{i=1}^{N_S} \Gamma_{ij} = 0$, the matrix $\boldsymbol{\Gamma}$ is singular and we have to modify it according to [24] as:

$$\widetilde{\boldsymbol{\Gamma}} = \boldsymbol{\Gamma} + \frac{\alpha}{\rho} \boldsymbol{Y} \otimes \boldsymbol{U}$$

where $\boldsymbol{U} = (1, ..., 1)^T$.

Since $\widetilde{\Gamma_{ij}} = \Gamma_{ij} + \frac{\alpha}{\rho} Y_i$ then $\sum_{i=1}^{N_S} \widetilde{\Gamma_{ij}} = \frac{\alpha}{\rho}\sigma$, hence we are able to determine the diffusive fluxes even in the presence of zero or vanishing mass fractions solving the linear system

$$\widetilde{\boldsymbol{\Gamma}} \boldsymbol{J} = -\boldsymbol{d}_i \tag{2.32}$$

In his work, Giovangigli developed also an iterative method to solve the aforementioned system; it is based on the following consideration: if we define $\boldsymbol{S} = \boldsymbol{I} - \boldsymbol{L}^{-1}\boldsymbol{\Gamma}$, with $\boldsymbol{L}$ to be determined later on, then $\boldsymbol{S}\boldsymbol{J} - \boldsymbol{L}^{-1}\boldsymbol{d}_i = \boldsymbol{J} + \boldsymbol{L}^{-1}\boldsymbol{d}_i - \boldsymbol{L}^{-1}\boldsymbol{d}_i = \boldsymbol{J}$.
Let us introduce the following vector space $\boldsymbol{U}^{\perp} = \left\{ \boldsymbol{x} \in \mathbb{R}^{N_S} : \boldsymbol{U} \cdot \boldsymbol{x} = 0 \right\}$ where

$$\boldsymbol{U} \cdot \boldsymbol{x} = \sum_{i=1}^{N_S} x_i.$$

There is only one solution $\boldsymbol{J}$ such that, given $\boldsymbol{d}_i \in \boldsymbol{U}^{\perp}$, it satisfies the conservation constraint; in particular the following theorem holds:

Theorem 1 *Let $\boldsymbol{\Gamma}$ be as in (2.28),(2.29), let $Y_i \geq 0, i = 1...N_S$ such that $\exists i \; Y_i \neq 0$ and let $\boldsymbol{L} = diag(L_1, ...L_{N_S})$ such that*

$$L_i = \frac{\Gamma_{ii}}{1 - \frac{Y_i}{\sigma}}$$

so that $L_i > \Gamma_{ii}$ if $Y_i > 0$ and $L_i \geq \Gamma_{ii}$ if $Y_i = 0$. Denote by $\boldsymbol{Q} = \boldsymbol{I} - \frac{\boldsymbol{Y} \otimes \boldsymbol{U}}{\sigma}$ and $\boldsymbol{S} = \boldsymbol{I} - \boldsymbol{L}^{-1}\boldsymbol{\Gamma}$ where $\boldsymbol{I}$ is the identity matrix; let $\boldsymbol{x_0} \in \mathbb{R}^{N_S}, \boldsymbol{y_0} = \boldsymbol{Q}\boldsymbol{x_0}$ and define

$$\boldsymbol{y}^{N+1} = \boldsymbol{Q}\left(\boldsymbol{S}\boldsymbol{y}^N - \boldsymbol{L}^{-1}\boldsymbol{d}_i\right).$$

Then

$$\boldsymbol{J} = \lim_{N\to\infty} \boldsymbol{y}^N$$

The drawback of this method is that the matrix $\boldsymbol{Q}$ projects each vector that belongs to $\mathbb{R}^{N_S}$ to $\boldsymbol{U}^\perp$; indeed we get:

$$[\boldsymbol{Q}\boldsymbol{x}]_i = x_i - \sum_{j=1}^{N_S} \frac{(\boldsymbol{Y}\otimes\boldsymbol{U})_{ij}}{\sigma} x_j = x_i - \sum_{j=1}^{N_S} \frac{Y_i}{\sigma} x_j = x_i - \frac{Y_i}{\sigma}\sum_{j=1}^{N_S} x_j$$

and therefore:

$$\begin{aligned}\sum_{i=1}^{N_S} [\boldsymbol{Q}\boldsymbol{x}]_i &= \sum_{i=1}^{N_S}\left(x_i - \frac{Y_i}{\sigma}\sum_{j=1}^{N_S} x_j\right) = \sum_{i=1}^{N_S} x_i - \sum_{i=1}^{N_S}\left(\frac{Y_i}{\sigma}\sum_{j=1}^{N_S} x_j\right) \\ &= \sum_{i=1}^{N_S} x_i - \sum_{j=1}^{N_S} x_j \sum_{i=1}^{N_S}\frac{Y_i}{\sigma} = \sum_{i=1}^{N_S} x_i - \sum_{j=1}^{N_S} x_j = 0\end{aligned}$$

Hence this iterative method is not able to compute the correct diffusive fluxes if $\boldsymbol{d}_i \notin \boldsymbol{U}^\perp$. Indeed in this case the scheme is not able to diffuse correctly errors in the chemical composition that, therefore, will be transported unchanged along the domain.

For this reason we choose to solve the linear system (2.32) with a "standard" iterative scheme, in particular the biconjugate gradient stabilized (**BiCGSTAB**) method (see Appendix A.7) because it does not need any particular requirement and this matches our case since the matrix $\widetilde{\boldsymbol{\Gamma}}$ is neither symmetric in general.

Eventually an appropriate choice for the free parameter α needed by the definition of $\widetilde{\boldsymbol{\Gamma}}$ is given by $\alpha = 1/\max_{i,j=1,\dots,N_S} D_{ij}$ because it guarantees the same order of magnitude for the elements of $\widetilde{\boldsymbol{\Gamma}}$.

The computational way to obtain the mole fractions gradient $\nabla X_i, i = 1, ..., N_S$ will be explained later on.

At this point we need to focus on the application of the mass flux constraint: the original Stefan-Maxwell equations are subjected to $\sum_{i=1}^{N_S} \boldsymbol{J}_i = 0$, but the use of artificial diffusivity changes this constraint; indeed summing up the relations (2.32) we find:

$$\alpha\sigma \sum_{i=1}^{N_S} \boldsymbol{J}_i = -\nabla\sigma\rho \tag{2.33}$$

and therefore we modify the continuity equation (2.6) to include this theoretical diffusion effect:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}) - \nabla\cdot\left(\frac{\rho}{\sigma\alpha}\nabla\sigma\right) = 0 \tag{2.34}$$

Hence the matrix associated to viscous fluxes in (2.17) is modified as follows:

$$\boldsymbol{G} = \begin{pmatrix} \frac{\rho}{\sigma\alpha}\nabla\sigma \\ \boldsymbol{\tau} \\ \boldsymbol{\tau u} - \boldsymbol{q} \\ -\boldsymbol{J}_1 \\ -\boldsymbol{J}_2 \\ \dots \\ -\boldsymbol{J}_{N_S} \end{pmatrix}$$

Let us notice finally that in case $\sigma \equiv 1$ everywhere equation (2.34) reduces to (2.6) and the constraint (2.33) reduces to $\sum_{i=1}^{N_S} \boldsymbol{J}_i = 0$ recovering the "*standard*" continuity equation.

Summing up, in this section we introduced a modification of Stefan-Maxwell equations which does not impose a priori that $\sum_{i=1}^{N_S} \boldsymbol{J}_i = \boldsymbol{0}$ but allows us to compute the diffusion fluxes in such a way that possible errors in the chemical composition due to the numerical method are diffused and dissipated.
In order to realize this process we need to consider also the "global" continuity equation as independent relation, so that $\sum_{i=1}^{N_S} Y_i = 1$ (or equivalently $\sum_{i=1}^{N_S} \rho_i = \rho$) is no more a constraint but a result of the algorithm, and moreover we need to modify it adding a diffusion term which takes into account the new version of Stefan-Maxwell equations.

2.3 Chemistry Source Term

A simple approach to obtain the chemical source term, as reported in [1], is the Perfectly Stirred Reactor(**PSR**) hypothesis proposed by [6] which considers chemical reactions in a pseudo-laminar condition:

$$\dot{\omega}_i = M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r} \right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho Y_j}{M_j} \right]^{\phi'_{j,r}} - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho Y_j}{M_j} \right]^{\phi''_{j,r}} \right\}, \quad i = 1...N_S \quad (2.35)$$

Here N_R denotes the number of reactions that the considered species partecipate in, $\nu'_{i,r}$ and $\nu''_{i,r}$ are the stoichiometric coefficients for species i in the reaction r as reactant and product respectively, while $\phi'_{j,r}$ and $\phi''_{j,r}$ are rate exponent for species j in reaction r as reactant and product.
Finally $k_{f,r}$ and $k_{b,r}$ denote the forward and backward rate constant for reaction r; the forward rate is computed using a generalized Arrhenius equation of the form

$$k_{f,r} = A_r T^{\beta_r} \exp(-E_r/RT)$$

where A_r is the exponential pre-factor which is derived experimentally, β_r is the temperature exponent and E_r is the activation energy. An alternative form uses the activation temperature T_r instead of the energy activation so that

$$k_{f,r} = A_r T^{\beta_r} \exp(-T_r/T)$$

The backward rate constant is computed using the relation

$$k_{b,r} = \frac{k_{f,r}}{K_r}$$

where K_r is the equilibrium constant of reaction r.

It is quite simple to show that the definition (2.35) guarantees that the constraint $\sum_{i=1}^{N_S} \dot{\omega}_i = 0$ is satisfied and therefore no source term appear to the global continuity

equation; indeed we get:

$$\begin{aligned}
\sum_{i=1}^{N_S}\dot{\omega}_i &= \sum_{i=1}^{N_S}\left\{M_i\sum_{r=1}^{N_R}\left(\nu_{i,r}^{''}-\nu_{i,r}^{'}\right)\left\{k_{f,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}}-k_{b,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}}\right\}\right\}\\
&= \sum_{i=1}^{N_S}\left\{\sum_{r=1}^{N_R}M_i\left(\nu_{i,r}^{''}-\nu_{i,r}^{'}\right)\left\{k_{f,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}}-k_{b,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}}\right\}\right\}\\
&= \sum_{r=1}^{N_R}\left\{\sum_{i=1}^{N_S}M_i\left(\nu_{i,r}^{''}-\nu_{i,r}^{'}\right)\left\{k_{f,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}}-k_{b,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}}\right\}\right\}\\
&= \sum_{r=1}^{N_R}\left\{\left\{k_{f,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}}-k_{b,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}}\right\}\sum_{i=1}^{N_S}M_i\left(\nu_{i,r}^{''}-\nu_{i,r}^{'}\right)\right\}
\end{aligned}$$

The term $\sum_{i=1}^{N_S}M_i\left(\nu_{i,r}^{''}-\nu_{i,r}^{'}\right)$ is equal to 0 for each reaction thanks to the balance due to stoichiometry and therefore $\sum_{i=1}^{N_S}\dot{\omega}_i=0$.

In general for non elementary reactions, i.e. reactions with more than one step, the rate exponents $\phi_{j,r}^{'}$ and $\phi_{j,r}^{''}$ that appear in (2.35) do not coincide with the stoichiometric coefficients $\nu_{i,r}^{'}$ and $\nu_{i,r}^{''}$ and therefore the effective computation of the source chemistry term is quite challenging.
Anyway if thanks to some experimental data we are able to know $\phi_{j,r}^{'}$ we can overcome the issue related to the computation of $\phi_{j,r}^{''}$ in the following manner; as reported in [31] at chemical equilibrium the following relations must simultaneously hold:

$$K_r=\frac{\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\nu_{j,r}^{''}}}{\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\nu_{j,r}^{'}}} \tag{2.36}$$

$$k_{f,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}}-k_{b,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}}=0 \tag{2.37}$$

The first relation comes from the definition of equilibrium constant while the second one derives from the fact that $\dot{\omega}_i=0$ at equilibrium. If we substitute (2.36) into (2.37) ed we recall that $k_{b,r}=\frac{k_{f,r}}{K_r}$ we find the following relation:

$$k_{f,r}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}}-k_{f,r}\frac{\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\nu_{j,r}^{'}}}{\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\nu_{j,r}^{''}}}\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}}=0$$

which implies

$$\prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{''}} = \prod_{j=1}^{N_S}\left[\frac{\rho Y_j}{M_j}\right]^{\phi_{j,r}^{'}-\nu_{j,r}^{'}+\nu_{j,r}^{''}} \tag{2.38}$$

and so:

$$\phi_{j,r}^{''} = \phi_{j,r}^{'} - \nu_{j,r}^{'} + \nu_{j,r}^{''} \qquad j = 1...N_S \tag{2.39}$$

Finally we need a way to compute the equilibrium constant K_r; since it is a thermodynamic constant, it exists a relation between its value and a thermodynamic quantity represented by **Gibbs free energy** G defined as

$$G = h - Ts$$

where h and s are the static enthalpy and the entropy respectively.
The aforementioned relation reads as follows:

$$\Delta G_r^{\circ} = -R_u T \ln(K_p) \tag{2.40}$$

where K_p is the equilibrium constant in terms of pressure and the symbol $\circ$ denotes the standard state pressure, i.e. $p_0 = 1$ atm.
Hess's law guarantees that the total change of Gibbs free energy during a reaction is independent of the pathway between the initial and final states: from a mathematical point of view this is translated by the following equation:

$$\Delta G_r^{\circ} = \sum \Delta^{\circ}_{G_{r_{products}}} - \sum \Delta^{\circ}_{G_{r_{reactants}}} \tag{2.41}$$

The relation 2.40 allows us to find the equilibrium constant in terms of pressure but, since we are interested to the equilibrium constant in terms of concentration, we need to apply the following relation:

$$K_r = K_p \left(\frac{R_u T}{p_{atm}}\right)^{-\Delta n_r} \tag{2.42}$$

where $\Delta n_r = \sum_{i=1}^{N_S}(\nu_{i,r}^{''} - \nu_{i,r}^{'})$ and p_{atm} denotes the atmospheric pressure.
It is worth to notice that the value $R_u = 8.31 \quad \mathrm{J\,mol^{-1}\,K^{-1}}$ is valid only if we use as units of the atmospheric pressure Pa(1 atm = 101325 Pa), while if we want to use directly the value 1 atm we need $R_u = 0.000\,082\,057\,338\,\mathrm{m^3\ atm\ K^{-1}\,mol^{-1}}$.

Chapter 3

Numerical Methods

The system of equations in (2.17) must be discretized both in space and in time in order to provide a numerical solution.

3.1 Space Integration

3.1.1 Vertex-Centered Finite Volume

During the last decades, the Finite Volume method [13, 14] has become one of the most employed techniques for simulating a wide variety of flows governed by hyperbolic equations.
The basic idea of this method is to subdivide the computational domain Ω into a disjoint set of finite cells or *volumes* and to apply inside each cell the conservation laws.
Let us introduce some useful notation: the division of Ω into N_C elements gives raise to the computational **mesh** or **grid** and the cell C_i is composed by a set of **vertices** V so that:

$$C_i, \qquad i = 1, ..., N_C$$

$$\Omega = \bigcup_{i=1}^{N_C} C_i$$

$$\mathring{C}_i \cap \mathring{C}_j = \emptyset, \qquad i, j = 1...N_C, i \neq j$$

Once a mesh has been formed, we have to create the finite volumes Ω_i on which the conservation laws will be applied. This can be done in two ways depending on where the solution is stored:

(1) If the solution is stored at the center of each C_i, then C_i itself is the finite volume, namely $\Omega_i = C_i$: this is the so called **cell-centered** finite volume method.

(2) If the solution is stored at the vertices of the mesh, then the finite volume Ω_i must be constructed around each vertex and this gives raise to the **vertex-centered** finite volume method.

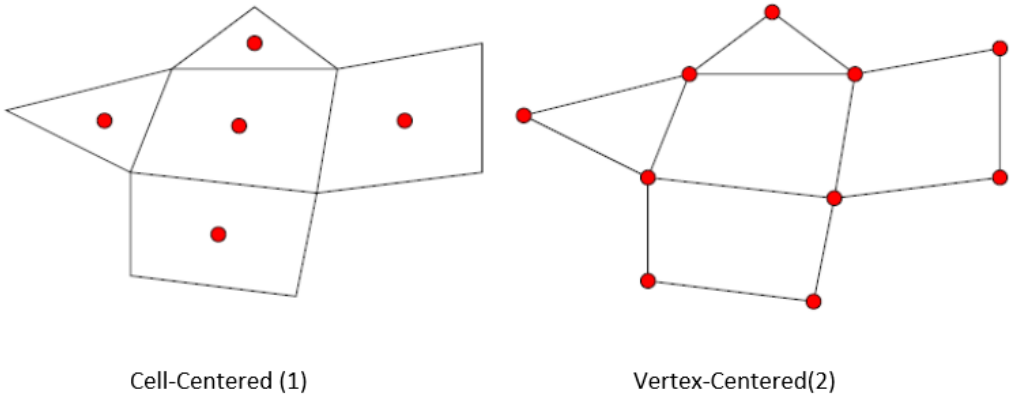


Figure 3.1: Finite volumes strategies

In either case we get a collection of finite volumes such that

$$\Omega = \bigcup_{i=1}^{N_C} \Omega_i$$

$$\mathring{\Omega}_i \cap \mathring{\Omega}_j = \emptyset, \qquad i, j = 1...N_C, i \neq j$$

The points where the solution is stored are called **nodes**.

The software that we will exploit adopts this second strategy and the finite volumes are formed by the centroids, face, and edge-midpoints of all cells sharing a particular node: their union is known as **dual grid** as shown is Figure (3.2).

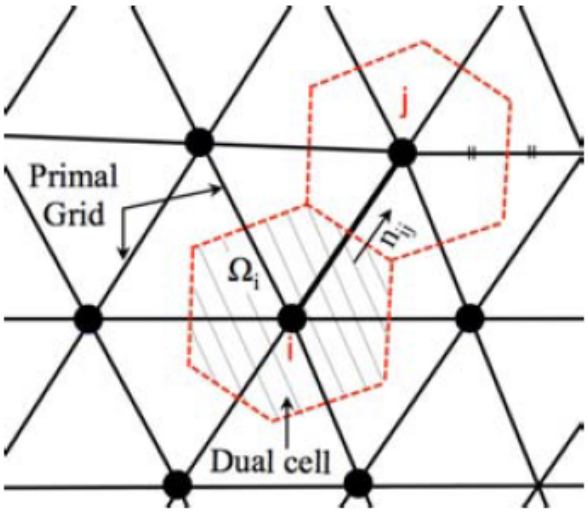


Figure 3.2: Schematic of the mesh and the control volume on a dual mesh.

The algorithm discretizes the system of PDE's in (2.17) written in an integral form:

$$\int_{\Omega_i} \frac{\partial \boldsymbol{Q}}{\partial t} d\Omega + \int_{\Omega_i} \nabla \cdot \boldsymbol{F} d\Omega - \int_{\Omega_i} \nabla \cdot \boldsymbol{G} d\Omega - \int_{\Omega_i} \boldsymbol{S} d\Omega = 0 \tag{3.1}$$

Now we exploit the divergence theorem to find:

$$\int_{\Omega_i} \frac{\partial \boldsymbol{Q}}{\partial t} d\Omega + \int_{\Sigma_i} \boldsymbol{F}\boldsymbol{n} d\Sigma - \int_{\Sigma_i} \boldsymbol{G}\boldsymbol{n} d\Sigma - \int_{\Omega_i} \boldsymbol{S} d\Omega = 0 \tag{3.2}$$

where Σ_i is the boundary of the finite volume Ω_i and $\boldsymbol{n}$ is the outward unit normal with respect to Σ_i. At this point we rewrite the equation (3.2) introducing the so called *residual* (or *steady residual*) so that

$$\int_{\Omega_i} \frac{\partial \boldsymbol{Q}}{\partial t} d\Omega + \boldsymbol{R_i}(\boldsymbol{Q}) = 0 \tag{3.3}$$

with

$$\boldsymbol{R_i}(\boldsymbol{Q}) = \int_{\Sigma_i} (\boldsymbol{F} - \boldsymbol{G})\, \boldsymbol{n} d\Sigma - \int_{\Omega_i} \boldsymbol{S} d\Omega \tag{3.4}$$

Every term in (3.4) is discretized separately leading to an upwind treatment of convective fluxes, a central discretization of diffusive fluxes and a vertex-centered treatment of the source term in combination with an explicit(forward Euler, Runge Kutta) or implicit scheme for the time stepping. Moreover second order accuracy in space can be obtained evaluating the fluxes with a linear polynomial reconstruction, while high order in time can be achieved by selecting a time integration scheme like $n - th$ order Runge-Kutta method.

3.1.2 Discretization of Convective Fluxes

The discretization of the convective term leads to:

$$\int_{\Sigma_i} \boldsymbol{F}\boldsymbol{n} d\Sigma = \sum_{f=1}^{N_f} \boldsymbol{F_f} \Sigma_f \tag{3.5}$$

where $\boldsymbol{F_f} = \widetilde{\boldsymbol{F_f}} \boldsymbol{n_f}$ is the numerical convective flux projected onto the normal $\boldsymbol{n_f}$ to the interface f with area Σ_f and N_f is the number of neighbour nodes of node i.

The numerical flux $\widetilde{\boldsymbol{F}}$ depends on the state vectors corresponding to the left and right neighboring points of the considered interface, namely:

$$\widetilde{\boldsymbol{F}} = \widetilde{\boldsymbol{F}}\left(\boldsymbol{Q}_i, \boldsymbol{Q}_j\right)$$

as reported in Figure (3.2).
The choice for multi-species reacting flows is a scheme of Advection Upstream Splitting Method (AUSM) family which is described in the next subsection.

AUSM Family

The schemes proposed by Liou and Steffen in [17–19] combine the efficiency of flux vector splitting methods with the accuracy and robustness of Godunov methods. The main idea is to split the projected numerical flux $\boldsymbol{F_f}$ into a convective term $\boldsymbol{F}^{(c)}$ and a pressure term $\boldsymbol{F}^{(p)}$ so that at a continuum level we find:

$$\boldsymbol{F_f} = \boldsymbol{F}^{(c)} + \boldsymbol{F}^{(p)} = \dot{m}\boldsymbol{\Psi} + \boldsymbol{F}^{(p)}$$

Herein $\boldsymbol{\Psi}$ is a vector quantity that represents the problem variables; in our context the mass flux $\dot{m}$, the vector $\boldsymbol{\Psi}$ and the pressure flux $\boldsymbol{F}^{(p)}$ can be therefore expressed as:

$$\dot{m} = \rho\boldsymbol{u}\cdot\boldsymbol{n}, \boldsymbol{\Psi} = \begin{pmatrix} 1 \\ \boldsymbol{u} \\ H \\ Y_i \end{pmatrix}, \boldsymbol{F}^{(p)} = \begin{pmatrix} 0 \\ p\boldsymbol{n} \\ 0 \\ 0 \end{pmatrix} \tag{3.6}$$

As we can see in (3.6) the convective term is constituted by a common scalar mass flux that takes into account the flow direction(the upwinding nature of AUSM) and problem variables which are convected by this mass flux, while the pressure term takes into account the contribution due only to pressure.

The discretization of the component of the numerical flux normal to a given interface depends on the left and right state vectors $\boldsymbol{Q}_L$ and $\boldsymbol{Q}_R$ and it can be defined as:

$$\boldsymbol{F}_{1/2}\left(\boldsymbol{Q}_L, \boldsymbol{Q}_R, \boldsymbol{n}\right) = \dot{m}_{1/2}\boldsymbol{\Psi}_{L/R} + \boldsymbol{p}_{1/2}$$

where $\boldsymbol{\Psi}_{L/R}$ will be determined in a simple upwind fashion,

$$\boldsymbol{\Psi}_{L/R} = \begin{cases} \boldsymbol{\Psi}_L, & \text{if } \dot{m}_{1/2} > 0 \\ \boldsymbol{\Psi}_R, & \text{otherwise} \end{cases}$$

Different choices for $\dot{m}_{1/2} = \dot{m}_{1/2}\left(\boldsymbol{Q}_L, \boldsymbol{Q}_R, \boldsymbol{n}\right)$ and $\boldsymbol{p}_{1/2} = \boldsymbol{p}_{1/2}\left(\boldsymbol{Q}_L, \boldsymbol{Q}_R, \boldsymbol{n}\right)$ determine different schemes: in this work the **AUSM$^+$-up** is chosen and therefore this method is briefly described in the next subsection.

AUSM$^+$-up

The formulation of this scheme is widely discussed in [19].
According to this method the mass flux scalar term $\dot{m}_{1/2}$ is a function of the interface Mach number, the left and right neighboring cells densities ρ_L, ρ_R and the interface sound speed $a_{1/2}$:

$$\dot{m}_{1/2} = M_{1/2}a_{1/2}\begin{cases} \rho_L, & \text{if } M_{1/2} > 0 \\ \rho_R, & \text{otherwise} \end{cases} \tag{3.7}$$

where $M_{1/2}$ is a polynomial function of left and right neighboring cells Mach numbers $\mathcal{M}_L$ and $\mathcal{M}_R$. The basic AUSM method in [17] defines:

$$M_{1/2}^{\text{AUSM}} = \mathcal{M}^+(M_L) + \mathcal{M}^-(M_R)$$

where the split Mach number polynomial $\mathcal{M}^\pm$ reads:

$$\mathcal{M}^\pm(M) = \begin{cases} \mathcal{M}^\pm_{(1)}(M), & \text{if } |M| > 1 \\ \mathcal{M}^\pm_{(2)}(M), & \text{otherwise} \end{cases}$$

with

$$\mathcal{M}^\pm_{(1)}(M) = \frac{1}{2}\left(M \pm |M|\right)$$

$$\mathcal{M}^\pm_{(2)}(M) = \pm\frac{1}{4}\left(M \pm 1\right)^2$$

and M_L and M_R are the normal left and right Mach numbers defined as:

$$M_L = \frac{\boldsymbol{u}_L \cdot \boldsymbol{n}}{a_{1/2}}$$

$$M_R = \frac{\boldsymbol{u}_R \cdot \boldsymbol{n}}{a_{1/2}}$$

The AUSM$^+$-up method redefines the split Mach numbers polynomials as:

$$\mathcal{M}^\pm_{(4)}(M) = \begin{cases} \mathcal{M}^\pm_{(1)}(M), & \text{if } |M| \geq 1 \\ \mathcal{M}^\pm_{(2)}(M)\left(1 \mp 16\beta\mathcal{M}^\mp_{(2)}(M)\right), & \text{otherwise} \end{cases}$$

where $-\frac{1}{16} \leq \beta \leq \frac{1}{2}$ In this way we can define $M_{1/2}^{\text{AUSM}^+\text{-up}} = \mathcal{M}^+_{(4)}(M_L) + \mathcal{M}^-_{(4)}(M_R)$.

Moreover the scheme adds a pressure diffusion term, namely:

$$M_{1/2} = M_{1/2}^{\text{AUSM}^+\text{-up}} - \frac{K_p}{f_a}\max\left(0, 1 - \sigma\bar{M}^2\right)\frac{p_R - p_L}{\rho_{1/2}a_{1/2}^2}$$

where $0 \leq K_p \leq 1$ and $\sigma \leq 1$. The interface density $\rho_{1/2}$ is defined as:

$$\rho_{1/2} = \frac{\rho_L + \rho_R}{2} \tag{3.8}$$

while the local mean Mach number $\bar{M}$ is such that:

$$\bar{M}^2 = \frac{1}{2}\left(M_L^2 + M_R^2\right)$$

At this point we can focus on the definition of the interface sound speed $a_{1/2}$; it can be expressed in several ways:

$$a_{1/2} = \sqrt{a_L a_R} \tag{3.9}$$

$$a_{1/2} = \frac{a_L + a_R}{2} \tag{3.10}$$

$$a_{1/2} = \min\left(\bar{a}_L, \bar{a}_R\right) \tag{3.11}$$

with

$$\bar{a}_{L/R} = \frac{a^{*2}}{\max\left(a^*, \left|\boldsymbol{u}_{L/R} \cdot \boldsymbol{n}\right|\right)}$$

where the critical velocity a^* expression is equal to:

$$a^* = \sqrt{\frac{2(\gamma-1)}{\gamma+1} H}$$

In this work the interface sound speed is computed using the simplest approach that corresponds to (3.10).

The same idea applies to the pressure flux term: for the standard AUSM scheme it has been defined as:

$$\boldsymbol{p}_{1/2}^{\text{AUSM}} = \mathcal{P}^+(M_L) p_L \boldsymbol{n} + \mathcal{P}^-(M_R) p_R \boldsymbol{n}$$

where the split polynomials $\mathcal{P}^{\pm}$ are given by:

$$\mathcal{P}^{\pm}(M) = \begin{cases} \frac{1}{M}\mathcal{M}^{\pm}_{(1)}(M), & \text{if } |M| > 1 \\ \pm\mathcal{M}^{\pm}_{(2)}(M)\left(2 \mp M\right), & \text{otherwise} \end{cases}$$

In the case of AUSM$^+$-up the split pressure polynomials $\mathcal{P}^{\pm}$ are modified as follows:

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

with $\frac{3}{16} \leq \alpha \leq \frac{1}{8}$.
In this way we can define $\boldsymbol{p}_{1/2}^{\text{AUSM}^+\text{-up}} = \mathcal{P}^+_{(5)}(M_L) p_L \boldsymbol{n} + \mathcal{P}^-_{(5)}(M_R) p_R \boldsymbol{n}$.
The pressure flux is finally modified adding an extra term like the mass flux and so we get:

$$\boldsymbol{p}_{1/2} = \boldsymbol{p}_{1/2}^{\text{AUSM}^+\text{-up}} - K_u \mathcal{P}^+_{(5)}(M_L)\mathcal{P}^-_{(5)}(M_R)\left(\rho_L + \rho_R\right)\left(f_a a_{1/2}\right)\left(\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u}_L \cdot \boldsymbol{n}\right)\boldsymbol{n}$$

with $0 \leq K_u \leq 1$ and the scaling factor f_a given by

$$f_a(M_0) = M_0\left(2 - M_0\right)$$

where

$$M_0^2 = \min\left(1, \max\left(\bar{M}^2, M_{co}^2\right)\right)$$

and the cut-off Mach number M_{co} is user-defined such that $M_{co} = O(M_\infty)$. Moreover the parameters α and β appearing in the split Mach and pressure functions are set to:

$$\alpha = \frac{3}{16}\left(5f_a^2 - 4\right) \qquad \beta = \frac{1}{8}$$

Eventually as suggested in [17, 18] we set as default values $K_p = 0.25$, $K_u = 0.75$ and $\sigma = 1.0$.
It is worth to notice the role of the scaling factor fa: as shown by Liou [19] it guarantees less diffusive and more robust scheme in case of low Mach numbers and on the other side its expression is such that $f_a = \mathcal{O}(1)$ if Ma $= \mathcal{O}(1)$.
Moreover, according to Liou, the method is **monotone** and positivity-preserving.
A scheme is said to be monotone if for two initial conditions u_j^0, v_j^0 with $u_j^0 \leq v_j^0$, then

$$u_j^n \leq v_j^n \qquad \forall n$$

while it is said to be positivity-preserving if given $u_j^n \geq 0$ then

$$u_j^{n+1}(x) \geq 0$$

These two properties guarantee the stability of the method.

3.1.3 Second Order Reconstruction

The AUSM scheme as presented so far assumes a constant average solution on each cell and this leads to a first order accurate discretization in space.

The idea to get second order accuracy is to linearly extrapolate the value of the variables denoted by $\boldsymbol{Q}_c$ at the centre of the cell to the face quadrature points q as follows:

$$\boldsymbol{Q}_q = \boldsymbol{Q}_c + \nabla \boldsymbol{Q}_c \left(\boldsymbol{x}_q - \boldsymbol{x}_c\right) \tag{3.12}$$

where $\boldsymbol{x}_c$ denotes the centroid position of the control volume Ω_c.
This extrapolation procedure is often indicated as **MUSCL** (Monotone Upstream-centered Schemes for Conservation Laws).
We choose to apply this reconstruction scheme on $T, \boldsymbol{u}$ and p, then the density is computed from the equation of state (A.2) leaving the mass fractions $Y_i, i = 1, ..., N_s$ fixed.
Now we need a procedure to compute the gradients needed in (3.12); in this work two different strategies have been employed: the **Green-Gauss** method or the **Least Squares** technique.
It is worth to notice that these procedures are employed also in the computation of $\nabla X_i, i = 1, ..., N_S$ for the numerical solution of Stefan-Maxwell equations.

Green-Gauss Method

Let us describe in this subsection the general procedure to determine the gradients of suitable variables $\boldsymbol{P}$ using Green-Gauss theorem.

The theorem applied to cell Ω_i reads as follows:

$$\int_{\Omega_i} \nabla \boldsymbol{P} d\Omega_i = \int_{\partial\Omega_i} \boldsymbol{P} \cdot \boldsymbol{n} d\Sigma_i$$

The discretized version therefore becomes:

$$\nabla \boldsymbol{P} = \frac{1}{|\Omega_i|} \sum_{f=1}^{N_f} \bar{\boldsymbol{P}}_f \boldsymbol{n}_f \Sigma_f$$

where $\bar{\boldsymbol{P}}_f$ is a face-averaged value of $\boldsymbol{P}$ computed from the values of neighbour nodes. The value of vector $\bar{\boldsymbol{P}}_f$ at the interface between two cells is computed through a simple arithmetic average: therefore if we denote by $\boldsymbol{P}_L$ and $\boldsymbol{P}_R$ the variables at left and right of the considered interface f respectively we get:

$$\bar{\boldsymbol{P}}_f = \frac{1}{2} \left(\boldsymbol{P}_L + \boldsymbol{P}_R \right)$$

Eventually in case of boundary cells it is pretty evident that we can not use average values but only the value on the corresponding node.

Least Squares Technique

Let us describe now the second approach to determine gradients: the least-squares technique. For the sake of simplicity we will describe the procedure for a scalar variable P_i: given data $P_1, P_2, P_3, ..., P_{\mathcal{N}_i}$ located at $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, ..., \boldsymbol{x}_{\mathcal{N}_i}$, the following Taylor expansion holds:

$$P_j = P_i + \nabla P_i \cdot (\boldsymbol{x_j} - \boldsymbol{x_i}) + \mathcal{O}(h^2), \qquad j = 1...\mathcal{N}_i$$

where $\mathcal{N}_i$ denotes the number of neighbour cells of node i.
Let us denote by $\nabla P_i = (a, b, c)^T$ the three components of the gradient and define:

$$\Delta P_j = P_j - P_i \qquad \Delta x_j = x_j - x_i \qquad \Delta y_j = y_j - y_i \qquad \Delta z_j = z_j - z_i$$

Neglecting higher order terms, we have on over-determined system of equations

$$\Delta P_j = \Delta x_j a + \Delta y_j b + \Delta z_j c$$

The components of ∇P_i are determined by solving the weighted minimization problem

$$\min_{a,b,c} \sum_{j=1}^{\mathcal{N}_i} \omega_j \left[\Delta P_j - \Delta x_j a - - \Delta y_j b - \Delta z_j c \right]^2$$

The weight function is chosen as:

$$\omega_j = \frac{1}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|^2}$$

Conditions for extrema are:

$$\frac{\partial}{\partial a}\sum_{j=1}^{\mathcal{N}_i}\omega_j\left[\Delta P_j-\Delta x_j a--\Delta y_j b-\Delta z_j c\right]^2=2\sum_{j=1}^{\mathcal{N}_i}\omega_j\left[-\Delta P_j\Delta x_j+(\Delta x_j)^2 a+\Delta x_j\Delta y_j b+\Delta x_j\Delta z_j c\right]=0$$

$$\frac{\partial}{\partial b}\sum_{j=1}^{\mathcal{N}_i}\omega_j\left[\Delta P_j-\Delta x_j a--\Delta y_j b-\Delta z_j c\right]^2=2\sum_{j=1}^{\mathcal{N}_i}\omega_j\left[-\Delta P_j\Delta y_j+\Delta y_j\Delta x_j a+(\Delta y_j)^2 b+\Delta y_j\Delta z_j c\right]=0$$

$$\frac{\partial}{\partial c}\sum_{j=1}^{\mathcal{N}_i}\omega_j\left[\Delta P_j-\Delta x_j a--\Delta y_j b-\Delta z_j c\right]^2=2\sum_{j=1}^{\mathcal{N}_i}\omega_j\left[-\Delta P_j\Delta z_j+\Delta z_j\Delta x_j a+\Delta z_j\Delta y_j b+(\Delta z_j)^2 c\right]=0$$

We obtained three coupled equations

$$\left(\sum_j\omega_j\Delta x_j^2\right)a+\left(\sum_j\omega_j\Delta x_j\Delta y_j\right)b+\left(\sum_j\omega_j\Delta x_j\Delta z_j\right)c=\sum_j\omega_j\Delta P_j\Delta x_j$$

$$\left(\sum_j\omega_j\Delta x_j\Delta y_j\right)a+\left(\sum_j\omega_j\Delta y_j^2\right)b+\left(\sum_j\omega_j\Delta z_j\Delta y_j\right)c=\sum_j\omega_j\Delta P_j\Delta y_j$$

$$\left(\sum_j\omega_j\Delta x_j\Delta z_j\right)a+\left(\sum_j\omega_j\Delta y_j\Delta z_j\right)b+\left(\sum_j\omega_j\Delta z_j^2\right)c=\sum_j\omega_j\Delta P_j\Delta z_j$$

which in matrix form becomes:

$$\begin{bmatrix}\sum_j\omega_j\Delta x_j^2 & \sum_j\omega_j\Delta x_j\Delta y_j & \sum_j\omega_j\Delta x_j\Delta z_j\\ \sum_j\omega_j\Delta x_j\Delta y_j & \sum_j\omega_j\Delta y_j^2 & \sum_j\omega_j\Delta z_j\Delta y_j\\ \sum_j\omega_j\Delta x_j\Delta z_j & \sum_j\omega_j\Delta y_j\Delta z_j & \sum_j\omega_j\Delta z_j^2\end{bmatrix}\begin{bmatrix}a\\b\\c\end{bmatrix}=\begin{bmatrix}\sum_j\omega_j\Delta P_j\Delta x_j\\ \sum_j\omega_j\Delta P_j\Delta y_j\\ \sum_j\omega_j\Delta P_j\Delta z_j\end{bmatrix}$$

We need at least two neighbouring cells to apply least squares method. In general it is better to provide a sufficiently large stencil in order to prevent singularities: here we chose the $\mathcal{N}_i$ face neighbouring cells which is usually sufficient to apply least squares successfully.
It is worth to note that the least squares method gives accurate gradient estimates, but on highly stretched grids it can lead to unstable schemes.

3.1.4 Flux Limiters

In order to prevent the appearance of spurious oscillations near shock waves, i.e. the so-called Gibbs phenomenon, when using a high order reconstruction, a flux limiter, here after indicated with $\mathbf{\Phi}$, is typically applied. Therefore the linear reconstruction specified in (3.12) must be modified as follows:

$$Q_{q_i}=Q_{c_i}+\Phi_{c_i}\cdot\nabla Q_{c_i}\cdot(\boldsymbol{x}_q-\boldsymbol{x}_c) \tag{3.13}$$

with $\Phi_{c_i}\in[0,1]\,\forall i$. This comes at a cost of an unavoidable accuracy deterioration in proximity of flow discontinuities and of a hampering of convergence.

Venkatakrishnan's Limiter

One common choice for the function $\mathbf{\Phi}_c$ is represented by the so called **Venkatakrishnan's limiter** [32].
Venkatakrishnan developed his multidimensional limiter in order to overcome some deficiencies of Barth-Jespersen's limiter [33], in particular the degradation of accuracy in nearly smooth flow regions.

Also in this case for the sake of simplicity the description is performed for a scalar variable Q_i and the corresponding limiter is denoted by Φ_i; in the original formulation the simple function $\Phi(r) = \min(r, 1)$ [33] is replaced by a differentiable function:

$$\Phi(r) = \frac{r^2 + r}{r^2 + r + 2}$$

and the limiter is given by:

$$\begin{cases} \Phi_i = \Phi(\frac{\Delta^+_{max}}{\Delta^-}), & \text{for } \Delta^- > 0 \\ \Phi_i = \Phi(\frac{\Delta^+_{min}}{\Delta^-}), & \text{for } \Delta^- < 0 \\ 1, & \text{for } \Delta^- = 0 \end{cases}$$

where

$$\Delta^- = \frac{1}{2}\Delta\boldsymbol{x} \cdot \nabla Q_i \qquad \Delta^+_{max} = \max_{j \in \mathcal{N}_i}(Q_i, Q_j) - Q_i \qquad \Delta^+_{min} = \min_{j \in \mathcal{N}_i}(Q_i, Q_j) - Q_i$$

where $\Delta\boldsymbol{x}$ is the vector that connects the node i with the one sharing the considered edge, while $\mathcal{N}_i$ denotes the set of neighbours of node i.

In order to maintain the accuracy in nearly uniform regions where $\Delta^+ \approx \Delta^- \approx 0$ the flux function is modified as follows:

$$\Phi\left(\frac{\Delta^+}{\Delta^-}\right) = \frac{{\Delta^+}^2 + 2\Delta^+\Delta^- + \delta}{{\Delta^+}^2 + \Delta^+\Delta^- + 2{\Delta^-}^2 + \delta}$$

where the δ term must scale to be negligible in non smooth flow regions and predominant where the flow is almost uniform. To this end δ can be estimated as

$$\delta = KU_0^2 \left(\frac{h}{L}\right)^3$$

where $U_0 \approx \mathcal{O}(\|\boldsymbol{u}\|)$ in the whole domain, h is a local characteristic length (such as the square root of current cell area), L is a local characteristic solution length in the smooth flow regions and $K \approx \mathcal{O}(1)$ is a user-defined constant.

The limiter is computed and applied separately for each variable in the solution vector.

3.1.5 Discretization of Diffusive Fluxes

The discretization of the viscous term gives raise to the following relation:

$$\int_{\Sigma_i} \boldsymbol{G}\boldsymbol{n} d\Sigma = \sum_{f=1}^{N_f} \boldsymbol{G_f} \Sigma_f \tag{3.14}$$

where $\boldsymbol{G_f} = \widetilde{\boldsymbol{G_f}}\boldsymbol{n_f}$ is the numerical diffusive flux projected onto the normal $\boldsymbol{n_f}$ to the interface f.
Typically it depends on the primitive variables $\boldsymbol{P}$ and on their gradients:

$$\widetilde{\boldsymbol{G}} = \widetilde{\boldsymbol{G}}\left(\boldsymbol{P}, \nabla \boldsymbol{P}\right)$$

Either the application of Green-Gauss theorem within a chosen control volume Ω^v or the least-squares technique as previously described can be used to determine the above mentioned gradients.

In our simulation we apply an **average gradient** approach with a suitable correction in order to reduce the truncation error of the scheme:

$$\nabla \boldsymbol{P} \cdot \boldsymbol{n} = \frac{1}{2}\left(\nabla \boldsymbol{P}|_i + \nabla \boldsymbol{P}|_j\right) \cdot \left(\boldsymbol{n}_{ij} - \alpha \boldsymbol{s}\right) + \frac{\boldsymbol{P_j} - \boldsymbol{P_i}}{\|\boldsymbol{x_j} - \boldsymbol{x_i}\|}\alpha \tag{3.15}$$

where $\boldsymbol{s}$ is the normalized vector connecting the cell centroid across the face, namely $\boldsymbol{s} = \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|}$, and $\alpha = \boldsymbol{s} \cdot \boldsymbol{n}_{ij}$.

3.1.6 Discretization of Source Term

Source terms are approximated using piecewise constant reconstruction within each of the finite volume cells, i.e.

$$\int_{\Omega_i} \boldsymbol{S} d\Omega \approx \boldsymbol{S}(\boldsymbol{P}_i)\Omega_i = \boldsymbol{S}_i \Omega_i \tag{3.16}$$

If the source term depends on some derivatives the Green-Gauss theorem or the least-squares technique can still be applied with control volume equal to the current cell.

3.2 Time Integration

The system of equations in (3.3) is an example of semidiscretazion of a system of PDEs; at this point we need a suitable discretization in time of (3.3) to obtain a numerical solution in space and in time. First of all let us reduce ourselves to a system of ODEs with the following semplification

$$\int_{\Omega_i} \frac{\partial \boldsymbol{Q}}{\partial t} d\Omega + \boldsymbol{R}_i\left(\boldsymbol{Q}\right) \approx \frac{d\boldsymbol{Q}_i}{dt}\left|\Omega_i\right| + \boldsymbol{R}_i\left(\boldsymbol{Q}\right) \tag{3.17}$$

where $|\Omega_i|$ is the volume of cell i.
Now we have several possibilities to discretize in time, for instance:

$$\frac{\boldsymbol{Q}_i^{n+1} - \boldsymbol{Q}_i^n}{\Delta t_i^n} + \boldsymbol{R}_i\left(\boldsymbol{Q}^n\right) = 0 \qquad \text{Explicit Euler (EE)} \tag{3.18}$$

$$\frac{\boldsymbol{Q}_i^{n+1} - \boldsymbol{Q}_i^n}{\Delta t_i^n} + \boldsymbol{R}_i\left(\boldsymbol{Q}^{n+1}\right) = 0 \qquad \text{Implicit Euler (IE)} \tag{3.19}$$

where the superscripts n and $n+1$ denote that the numerical solutions are evaluated at step n and $n+1$ respectively.
Δt_i^n is the time step for the cell i at time n: indeed **local-time stepping** strategies are applied so that each volume can advance at a different time step according to the local values of the variables of the problem. The following definition applies:

$$\Delta t_i = N_{CFL} \min\left(\frac{|\Omega_i|}{\lambda_i^{conv}}, \frac{|\Omega_i|}{\lambda_i^{visc}}\right)$$

where N_{CFL} is the Courant-Friedrichs-Lewy (CFL) number and λ_i^{conv} is the integrated convective spectral radius computed as

$$\lambda_i^{conv} = \sum_{f=1}^{N_f} \left(\left|u_{1/2}\right| + c_{1/2}\right) \Sigma_f$$

where $\left|u_{1/2}\right|$ is the absolute value of the interface velocity computed as $\left|\frac{\boldsymbol{u}_L + \boldsymbol{u}_R}{2} \cdot \boldsymbol{n}_f\right|$ and $c_{1/2}$ is the interface sound speed compute as (3.9). On the other hand the viscous spectral radius λ_i^{visc} is computed as:

$$\lambda_i^{visc} = \sum_{f=1}^{N_f} \frac{C\mu_{1/2} + f(\mu_{1/2}^L + \mu_{1/2}^T)}{\rho_{1/2}} \Sigma_f^2$$

Here $\rho_{1/2}$ is the interface density already defined in (3.8), C is a constant, $\mu_{1/2}$ is the interface viscosity defined as the sum of the interface laminar viscosity $\mu_{1/2}^L = \frac{\mu_L^L + \mu_R^L}{2}$ and the interface eddy viscosity $\mu_{1/2}^T = \frac{\mu_L^T + \mu_R^T}{2}$ in case of a turbulent simulation and f is a suitable function defined as

$$f(\mu_{1/2}^L + \mu_{1/2}^T) = \left(1.0 + \frac{Pr_{1/2}^L}{Pr_{1/2}^T} \frac{\mu_{1/2}^T}{\mu_{1/2}^L}\right) \left(\gamma_{1/2} \frac{\mu_{1/2}^L}{Pr_{1/2}^L}\right) \tag{3.20}$$

where $Pr_{1/2}^L$ and $Pr_{1/2}^T$ are the laminar and turbulent Prandtl number respectively defined as:

$$Pr_{1/2}^{L/T} = \frac{\mu_{1/2}^{L/T} C_{p_{1/2}}}{\kappa_{1/2}^{L/T}}$$

with $C_{p_{1/2}} = \frac{C_{p_L}+C_{p_R}}{2}$ as interface specific heat at constant pressure and $\kappa_{1/2}^{L/T} = \frac{\kappa_L^{L/T}+\kappa_R^{L/T}}{2}$ as laminar and turbulent interface thermal conductivity.
Finally $\gamma_{1/2} = \frac{C_{p_{1/2}}}{C_{v_{1/2}}}$ where $C_{v_{1/2}} = \frac{C_{v_L}+C_{v_R}}{2}$ is the interface specific heat at constant volume.

In case of Explicit Euler scheme the solution update $\Delta \boldsymbol{Q}_i^n = \boldsymbol{Q}_i^{n+1} = \boldsymbol{Q}_i^n$ is immediately found as:

$$\Delta \boldsymbol{Q}_i^n = -\boldsymbol{R}_i(\boldsymbol{Q}^n)\Delta t_i^n \tag{3.21}$$

while in case of Implicit Euler scheme the residuals at time $n+1$ are unknown and therefore a linearization about t^n is needed:

$$\boldsymbol{R}_i(\boldsymbol{Q}^{n+1}) = \boldsymbol{R}_i(\boldsymbol{Q}^n) + \frac{\partial \boldsymbol{R}_i(\boldsymbol{Q}^n)}{\partial t}\Delta t^n + \mathcal{O}(\Delta t^2) = \boldsymbol{R}_i(\boldsymbol{Q}^n) + \sum_{j=1}^{N_f} \frac{\partial \boldsymbol{R}_i(\boldsymbol{Q}^n)}{\partial \boldsymbol{Q}_j^n}\Delta \boldsymbol{Q}_j^n + \mathcal{O}(\Delta t^2)$$

Finally the following linear system should be solved to find the solution update $\Delta \boldsymbol{Q}_i^n$:

$$\left(\frac{|\Omega_i|}{\Delta t_i^n}\delta_{ij} + \frac{\partial \boldsymbol{R}_i(\boldsymbol{Q}^n)}{\partial \boldsymbol{Q}_j^n}\right)\Delta \boldsymbol{Q}_j^n = -\boldsymbol{R}_i(\boldsymbol{Q}^n)$$

The term $\frac{\partial \boldsymbol{R}_i(\boldsymbol{Q}^n)}{\partial \boldsymbol{Q}_j^n}$ is constituted by the contribution of convective numerical flux Jacobian (see Appendix A.4), viscous numerical flux Jacobian(see Appendix A.5) and source term Jacobian (see Appendix A.6) and in case the total flux $\tilde{F}_{ij}$ has a stencil of points $\{i,j\}$, then contributions are made to the Jacobian at four points:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}} = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \frac{\partial \tilde{F}_{ij}}{\partial \boldsymbol{Q}_i} & \cdots & \frac{\partial \tilde{F}_{ij}}{\partial \boldsymbol{Q}_j} & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \cdots & \cdots & -\frac{\partial \tilde{F}_{ij}}{\partial \boldsymbol{Q}_i} & \cdots & -\frac{\partial \tilde{F}_{ij}}{\partial \boldsymbol{Q}_j} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Finally we set $\boldsymbol{Q}^{n+1} = \boldsymbol{Q}^n + \Delta \boldsymbol{Q}^n$.

The software allows also the use of a **dual time-stepping** strategy in order to achieve high-order accuracy in time. The main idea of this method is to transform an unsteady problem into a steady one at each physical time step; therefore the implementation of the dual-time stepping approach solves the following problem:

$$\frac{\partial \boldsymbol{Q}}{\partial \tau} + \boldsymbol{R}^*(\boldsymbol{Q}) = 0$$

with

$$\boldsymbol{R}^*(\boldsymbol{Q}) = \frac{3}{2\Delta t}\boldsymbol{Q} + \frac{1}{|\Omega|^{n+1}}\left(\boldsymbol{R}(\boldsymbol{Q}) - \frac{2}{\Delta t}\boldsymbol{Q}^n|\Omega|^n + \frac{1}{2\Delta t}\boldsymbol{Q}^{n-1}|\Omega|^{n-1}\right)$$

where Δt is the physical time step and τ is a fictitious time step used for the convergence of the steady problem: therefore we set $\boldsymbol{Q} = \boldsymbol{Q}^{n+1}$ once the steady problem is satisfied. Obviously in case the control volume is fixed $\left|\Omega^{n-1}\right| = |\Omega^n| = \left|\Omega^{n+1}\right|$

3.3 Boundary Conditions

A set of boundary conditions is necessary to solve the governing equations (2.17) at each time step. For node-centric finite volume solvers the solution is stored directly on the computational boundary: this allows boundary conditions to be enforced either weakly or strongly. In weak enforcement, the governing equations are written on the boundary and a flux is computed such that when the solution achieves convergence, the condition is satisfied, while in strong enforcement Dirichlet conditions are set for one or more scalar variables at boundary and any contribution to the solution residual or Jacobian from flux calculations is eliminated to preserve the specified boundary condition. The main required boundary conditions and their numerical implementation are described in this section.

3.3.1 Euler Wall

It requires the local velocity to align with the boundary and it is used to represent objects in inviscid flow environments or to represent planes of symmetry in the domain. Explicitly the boundary condition is stated as

$$\boldsymbol{u} \cdot \boldsymbol{n} = 0$$

This flow tangency boundary condition is weakly enforced by imposing the numerical flux

$$\hat{\boldsymbol{F}}_b = \begin{pmatrix} 0 \\ p\boldsymbol{n} \\ 0 \\ 0 \\ 0 \\ .... \\ 0 \end{pmatrix}$$

over the slip-wall faces associated with boundary b.

3.3.2 No-Slip Isothermal Wall

The no slip condition which requires the flow velocity at wall equal to zero is strongly imposed, while for what concerns the isothermal condition we impose a numerical flux

equal to:

$$\hat{\boldsymbol{F}}_{iso} = \begin{pmatrix} 0 \\ \mathbf{0} \\ \frac{\kappa(T_{int}-T_w)}{d_{ij}} \\ 0 \\ 0 \\ .... \\ 0 \end{pmatrix}$$

where T_{int} is the temperature at the nearest internal node, T_w is the prescribed temperature and d_{ij} is the distance between the nearest internal node and the boundary node. As it is evident from the flux expression we employ a simple finite difference scheme to approximate the heat flux due to temperature gradient and we treat the wall as non-catalytic, i.e. $\nabla Y_i \cdot \boldsymbol{n} = 0, i = 1...N_S$.

3.3.3 Subsonic Inlet

More attention is needed for what concerns the inlet boundary conditions; two cases must be distinguished: supersonic and subsonic inlet.
In both cases the **Riemann invariants** have to be taken into account [27].
Starting from a generic system of conservation laws:

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial x} = \mathbf{0}$$

we can express it in the semilinear form

$$\frac{\partial \boldsymbol{q}}{\partial t} + \boldsymbol{A}\frac{\partial \boldsymbol{q}}{\partial x} = \mathbf{0}$$

Moreover

$$\boldsymbol{A} = \boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^{-1}$$

where $\boldsymbol{\Lambda} = diag(\lambda_1, ..., \lambda_m)$ and the columns of $\boldsymbol{\Gamma}$ are the right eigenvectors of $\boldsymbol{A}$.
If we set $\boldsymbol{p} = \boldsymbol{\Gamma}^{-1}\boldsymbol{q}$, we find:

$$\frac{\partial \boldsymbol{p}}{\partial t} + \boldsymbol{\Lambda}\frac{\partial \boldsymbol{p}}{\partial x} = \mathbf{0}$$

In the previous relation, the single equations are decoupled and assume the form:

$$\frac{\partial p_k}{\partial t} + \lambda_k \frac{\partial p_k}{\partial x} = 0, \qquad k = 1, ..., m$$

It is natural to define the **characteristic curve** as solutions of the equations:

$$\frac{dx}{dt} = \lambda_k(x, t), \qquad k = 1, ..., m$$

and a **Riemann invariant** is a scalar quantity which is constant along the characteristic curve.

In general a regular function $w_k : \mathbb{R}^m \to \mathbb{R}$ is called **k-Riemann invariant** if it satisfies the following equation:

$$\nabla w_k(\boldsymbol{q}) \cdot \boldsymbol{r}_k(\boldsymbol{q}) = 0$$

where $\boldsymbol{r}_k$ is the $k-$th right eigenvector of matrix $\boldsymbol{A}$.
In case of supersonic inlet there are no outgoing characteristics and therefore all variables can be directly specified at inlet, while in case of subsonic inlet there is one outgoing characteristic. Usually the Riemann invariant associated to it is expressed as

$$V_n + 2\frac{c}{\gamma - 1} = const \tag{3.22}$$

where V_n is the normal velocity, but unluckily this relation holds only considering the isentropic relations for ideal gas, i.e. $\gamma = const$, as specified in [34]

$$\frac{p}{\rho^\gamma} = const \tag{3.23}$$

In our case, since we are considering thermally perfect gases and γ is not constant, we can not deal a priori to this relation. One possible novel approach can be starting from differential relations and then, when no exact integral is available, approximate it through suitable quadrature formulas.

In this analysis we will focus on the so called **total inlet boundary conditions** which impose the velocity, the **total temperature** and the **total pressure**, i.e. the temperature and the pressure when the fluid is at rest.
In order to pass from the "total" state to the local boundary state we impose the conditions of **isentropicity** and **adiabaticity**.
Let us express the two aforementioned conditions and the Riemann invariant in differential form:

$$\begin{cases} C_v \dfrac{dT}{T} - R\dfrac{d\rho}{\rho} = 0 & (3.24) \\ dh + VdV = 0 & (3.25) \\ dV_n + \dfrac{c}{\rho} d\rho = 0 & (3.26) \end{cases}$$

where the subscripts b and i denote the boundary and inner state respectively. If we develop and we start integrating we get:

$$\begin{cases} \int_{T_{tot}}^{T_b} C_p dT - \int_{T_{tot}}^{T_b} R_b dT - \int_{T_{tot}}^{T_b} R_b \frac{T(\rho)}{\rho} d\rho = 0 \\ \int_{H_{tot}}^{h_b} dh + \int_0^{V_b} V dV = 0 \\ \int_{V_{n,i}}^{V_{n,b}} dV_n + \int_{\rho_i}^{\rho_b} \frac{c}{\rho} d\rho = 0 \end{cases}$$

where H_{tot} is the total enthalpy computed at the imposed total temperature T_{tot}.
It is worth to notice that (3.24) and (3.25) are integrated between "total" and "local" boundary conditions, while (3.26) is integrated between boundary and inner states.

Moreover R_b is the mixture gas constant computed with the composition imposed at inlet and since we assume that mass fractions do not change between "total" and "local" state, R_b is constant along the integration.

We obtain:

$$\begin{cases} \int_{T_{tot}}^{T_b} C_p dT - R_b \left(T_b - T_{tot}\right) - \int_{T_{tot}}^{T_b} R_b \frac{T}{\rho} d\rho = 0 \\ h_b + \frac{V_b^2}{2} = H_{tot} \\ V_{n,b} + \int_{\rho_i}^{\rho_b} \frac{c}{\rho} d\rho = V_{n,i} \end{cases} \tag{3.27}$$

Applying the trapezoidal rule we find:

$$\begin{cases} h_b - H_{tot} - R_b(T_b - T_{tot}) - R_b \left(\frac{\rho_b - \rho_{tot}}{2}\right) \left[\frac{T_b}{\rho_b} + \frac{T_{tot}}{\rho_{tot}}\right] = 0 & (3.28) \\ h_b + \frac{V_b^2}{2} = H_{tot} & (3.29) \\ V_b \alpha + \left(\frac{\rho_b - \rho_i}{2}\right) \left[\frac{c_b}{\rho_b} + \frac{c_i}{\rho_i}\right] = V_{n,i} & (3.30) \end{cases}$$

where α is the inner product between the inlet velocity direction and the outward unit normal and ρ_{tot} is computed using the gas equation with T_{tot} and the total pressure p_{tot}.

Let us derive V_b from (3.30):

$$\begin{aligned} V_b &= \frac{1}{\alpha} \left[V_{n,i} - \left(\frac{\rho_b - \rho_i}{2}\right) \left(\frac{c_b}{\rho_b} + \frac{c_i}{\rho_i}\right)\right] \\ &= \frac{1}{\alpha} \left[V_{n,i} + \frac{1}{2} \left(\rho_i \frac{c_b}{\rho_b} + c_i - \rho_b \frac{c_i}{\rho_i} - c_b\right)\right] = V_b(\rho_b(T_b)) \end{aligned} \tag{3.31}$$

Now we need to express $\rho_b = \rho_b(T_b)$ in order then to substitute into (3.29) and apply an iterative scheme like the secant method (see Appendix A.9) to solve the implicitly defined relation; from (3.28) we find:

$$\begin{aligned} & h_b - H_{tot} - R_b(T_b - T_{tot}) - \frac{R_b}{2} \left(T_b + \rho_b \frac{T_{tot}}{\rho_{tot}} - \rho_{tot} \frac{T_b}{\rho_b} - T_{tot}\right) = 0 \implies \\ & h_b - H_{tot} - \frac{3}{2} R_b \left(T_b - T_{tot}\right) - \frac{R_b}{2} \rho_b \frac{T_{tot}}{\rho_{tot}} + \frac{R_b}{2} \rho_{tot} \frac{T_b}{\rho_b} = 0 \implies \\ & - \frac{R_b}{2} \frac{T_{tot}}{\rho_{tot}} \rho_b^2 + \left[n_b - H_{tot} - \frac{3}{2} R_b (T_b - T_{tot})\right] \rho_b + \frac{R_b}{2} \rho_{tot} T_{tot} = 0 \implies \\ & \frac{R_b}{2} \frac{T_{tot}}{\rho_{tot}} \rho_b^2 - \left[n_b - H_{tot} - \frac{3}{2} R_b (T_b - T_{tot})\right] \rho_b - \frac{R_b}{2} \rho_{tot} T_{tot} = 0 \end{aligned} \tag{3.32}$$

The relation (3.32) is a second degree equation whose only physical solution is

$$\rho_b = \frac{\left[h_b - H_{tot} - \frac{3}{2} R_b (T_b - T_{tot})\right] + \sqrt{\left[h_b - H_{tot} - \frac{3}{2} R_b (T_b - T_{tot})\right]^2 + R_b^2 T_{tot} T_b}}{\frac{R_b T_{tot}}{\rho_{tot}}} = \rho_b(T_b) \tag{3.33}$$

Eventually if we substitute (3.33) into (3.31) and (3.31) into (3.29) we get:

$$h_b(T_b) + \frac{V_b(\rho_b(T_b))^2}{2} = H_{tot} \tag{3.34}$$

We underline the fact that since we use Burcat polynomials to compute the enthalpy as we will see later on, we still have to rely on **secant method** to determine the boundary temperature at inlet through (3.34) because temperature is defined by enthalpy only implicitly. (see Appendix A.9).

Eventually it is worth to notice that the choice of trapezoidal rule has been due to the fact that we need to derive explicitly ρ_b as a function of T_b but more accurate quadrature rules can be applied.

3.3.4 Subsonic outlet

An analogous discussion holds also for the outlet boundary condition: in case of supersonic outlet there are no incoming characteristics and therefore all the variables can be extrapolated from the interior state, while in case of subsonic outlet there is one incoming characteristics and therefore the value of one variable depends on the state outside the computational domain.
The typical choice, applied also in this work, is to impose the static back pressure p_b at outlet boundary and then to rely on isentropicity between the outer and the inner state in order to update the boundary variables.
As stated in [34], the isentropicity condition can be also expressed in differential form as:

$$C_p \frac{dT}{T} - R\frac{dp}{p} = 0 \tag{3.35}$$

which can be integrated:

$$\int_{T_i}^{T_b} \frac{C_p}{T} dT - R \ln\left(\frac{p_b}{p_i}\right) = 0 \tag{3.36}$$

where R is constant because we choose to keep on the boundary the same mass fractions of the inner node.
Now we need to approximate the integral that appears in (3.36) using a quadrature formula; herein we choose to employ the so-called Cavalieri-Simpson rule which represents a good compromise between accuracy and computational cost. Therefore we get:

$$\int_{T_i}^{T_b} \frac{C_p}{T} dT = \frac{T_b - T_i}{6}\left[\frac{C_p(T_i)}{T_i} + 4\frac{C_p(\frac{T_i+T_b}{2})}{\frac{T_i+T_b}{2}} + \frac{C_p(T_b)}{T_b}\right] = f(T_b) \tag{3.37}$$

from which we find the following implicit equation:

$$F(T_b) = f(T_b) - R \ln\left(\frac{p_b}{p_i}\right) = 0 \tag{3.38}$$

which can be solved iteratively through the secant method.
Up to now we did not mention anything about the information from the Riemann invariant: now it's time to exploit the incoming characteristic in order to find the boundary velocity; indeed the aforementioned Riemann invariant can be stated in the form:

$$V_{b,n} - \int_{\rho_i}^{\rho_b} \frac{c}{\rho} d\rho = V_{i,n} \tag{3.39}$$

The integral in (3.39) can be approximated through quadrature rules: herein we choose a three-point Gaussian quadrature formula; since we can map the interval between ρ_i and ρ_b (determined through (2.5)) exploiting the isentropicity, this scheme represents a good compromise between accuracy and computational cost.
More in detail we get:

$$\int_{\rho_i}^{\rho_b} \frac{c(T)}{\rho} d\rho = \int_{\rho_i}^{\rho_b} f(\rho) d\rho \approx \frac{\rho_b - \rho_i}{2} \sum_j w_j f\left(\frac{\rho_b - \rho_i}{2}\psi_j + \frac{\rho_b + \rho_i}{2}\right) = \frac{\rho_b - \rho_i}{2} \sum_j w_j f(\tilde{\psi}_j) \tag{3.40}$$

with

$$w_j = \begin{bmatrix} \frac{5}{9} & \frac{8}{9} & \frac{5}{9} \end{bmatrix} \qquad \psi_j = \begin{bmatrix} -\frac{\sqrt{3}}{5} & 0 & \frac{\sqrt{3}}{5} \end{bmatrix}$$

The intermediate temperature $\tilde{T}_j$ at corresponding density $\tilde{\rho}_j$ to compute the speed of sound is determined through the isentropicity, as stated before:

$$\int_{\tilde{T}_{j-1}}^{\tilde{T}_j} \frac{C_p}{T} dT - R \int_{\tilde{T}_{j-1}}^{\tilde{T}_j} \frac{dT}{T} - R \int_{\tilde{\psi}_{j-1}}^{\tilde{\psi}_j} \frac{d\rho}{\rho}$$

As it can be easily noticed, in the previous discussion we applied two different quadrature formula for resolving subsonic inlet and outlet. The reason is that the first boundary condition requires lower order rules if we want to obtain relatively easily an analytic expression, while the second one can be manipulated even in case of higher order formulas. In any case it is worth to notice that (3.27) can be treated as a non-linear system of equations where the integrals can be approximated with any kind of quadrature rule and then the system is solved with a suitable scheme such as Newton's method.
Analogous considerations hold also for (3.36).

Anyway both this approach for subsonic outlet and the one previously described for subsonic inlet do not guarantee a significant improvement in the accuracy of boundary condition especially if compared with the computational cost: indeed we run out various simulations with random generated inner states and we checked that time required grows up more than 50 times as we can see from the results of one of them

	Approximation $\gamma = const$	My algorithm
$V_{n,b}$	147.226900273	147.2501327862
T_b	3433.8045	3433.7451
ρ_b	0.0194	0.0194
c_b	2801.0706	2800.1045
Execution Time	312.8 ns	16274.1 ns

Table 3.1: Comparison between two approaches for subsonic outlet boundary conditions ($p_i = 101824.4595, T_i = 3365.4756, p_b = 109368.0710$)

The relative error for the temperature is $1.7307 \cdot 10^{-5}$ which does not justify a so relevant increase in the time needed.

Therefore we choose to pick $\gamma = const$ approximation taking the internal value for the subsonic outlet and the harmonic average between the internal value and the total one in order to consider the contribution of both states for the subsonic inlet and therefore we assume the validity of (3.23).
We set

$$R^+ = V_{n,i} + \frac{2c_i}{\gamma - 1}$$

In case of subsonic inlet we compute V_b as

$$V_b = \frac{R^+ - \frac{2c_b(T)}{\gamma - 1}}{\alpha}$$

which we substitute into (3.34) to find the temperature. Then we employ the state equation to complete properly the state

In case of subsonic outlet we compute the density from (3.23) and consequently the speed of sound, while the normal velocity $V_{n,b}$ is found as

$$V_{n,b} = R^+ - \frac{2c_b}{\gamma - 1}$$

Eventually we apply the state equation to find the temperature.

For further detail, please refer on [37].

Chapter 4

SU2 Code

The system of equations previously introduced is implemented in the **SU2** suite, an open-source collection of C++ based software tools for performing Partial Differential Equations(PDE) analysis and solving PDE-constrained optimization problems.
For a more detailed analysis, please refer to [2]

4.1 General Features

At the highest level, SU2 has a driver class, **CDriver**, that controls the solution of a multiphysics simulation. The CDriver class is responsible for instantiating all of the geometry, physics packages, and numerical methods needed to solve a particular problem as we can see in (4.1). In particular the constructor of this class calls all the routine

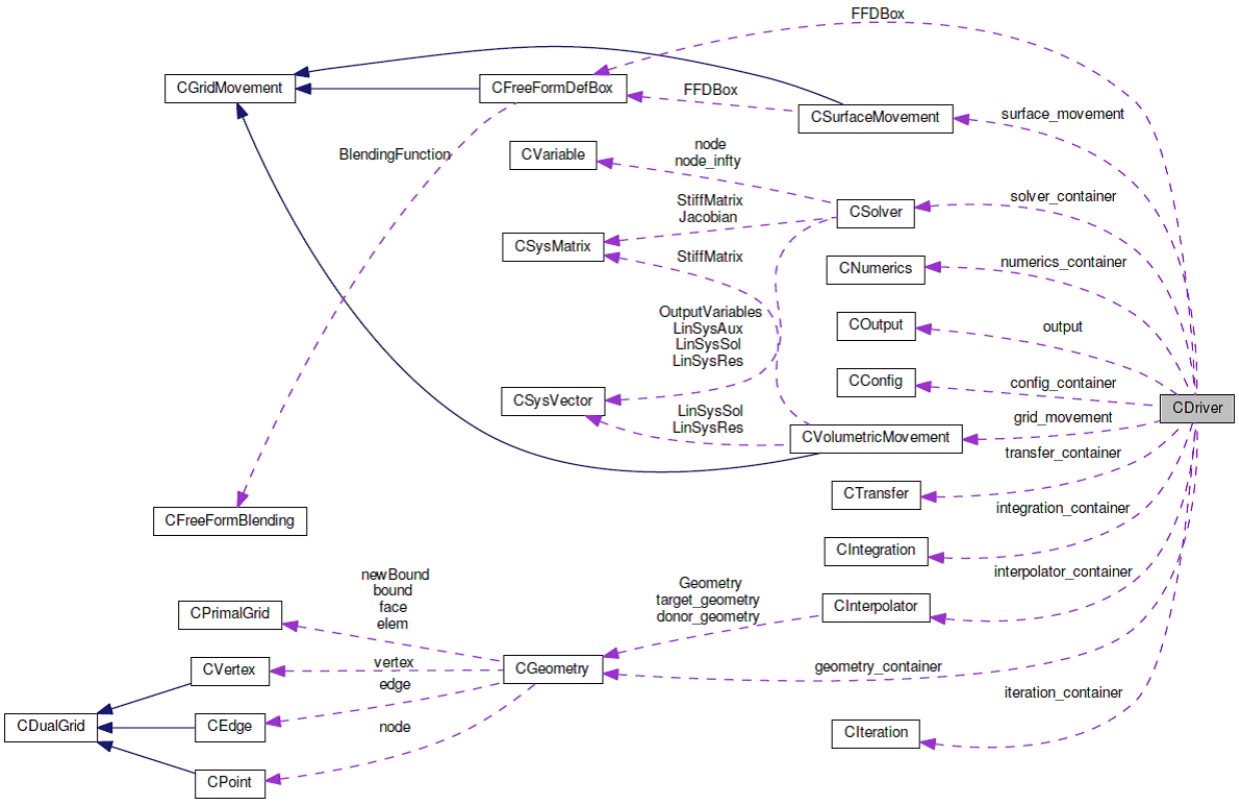


Figure 4.1: General structure of SU2

involved to the preprocessing of the simulation that ensure the physical reliability of data.

```
if (rank == MASTER_NODE)
  cout << endl <<"------------------------- Geometry Preprocessing
      -------------------------" << endl;
Geometrical_Preprocessing();
...

if (rank == MASTER_NODE) {
  cout << endl <<"------------------------- Iteration Preprocessing
      -------------------------" << endl;
}
Iteration_Preprocessing();
...

if (rank == MASTER_NODE)
  cout << endl <<"------------------------- Solver Preprocessing
      --------------------------" << endl;

solver_container[iZone] = new CSolver**
    [config_container[iZone]->GetnMGLevels()+1];
for (iMesh = 0; iMesh <= config_container[iZone]->GetnMGLevels(); iMesh++)
  solver_container[iZone][iMesh] = NULL;

  for (iMesh = 0; iMesh <= config_container[iZone]->GetnMGLevels();
      iMesh++) {
    solver_container[iZone][iMesh] = new CSolver* [MAX_SOLS];
    for (iSol = 0; iSol < MAX_SOLS; iSol++)
      solver_container[iZone][iMesh][iSol] = NULL;
  }
Solver_Preprocessing(solver_container[iZone], geometry_container[iZone],
config_container[iZone]);
...

if (rank == MASTER_NODE)
  cout << endl <<"----------------- Integration and Numerics Preprocessing
      ----------------" << endl;

integration_container[iZone] = new CIntegration*[MAX_SOLS];
Integration_Preprocessing(integration_container[iZone],
    geometry_container[iZone],
config_container[iZone]);

if (rank == MASTER_NODE) cout << "Integration Preprocessing." << endl;

numerics_container[iZone] = new
    CNumerics***[config_container[iZone]->GetnMGLevels()+1];
Numerics_Preprocessing(numerics_container[iZone], solver_container[iZone],
geometry_container[iZone], config_container[iZone]);

if (rank == MASTER_NODE) cout << "Numerics Preprocessing." << endl;
```

4.2 Additions

For our purposes, since we are interested in the implementation of a new kind of problem, we have to focus on the classes which represent the interface for physical problems which are **CVariable**, **CSolver** and **CNumerics**.
Therefore we need to implement the numerical methods described before in order to compute the fluxes with distinction between convective, diffusive and source term: hence we built three different classes all derived by the base class **CNumerics**.
Secondly we need to store opportunely the physical state of a multispecies simulation and we created two classes (**CReactiveEulerVariable** and **CReactiveNSVariable**) which enriches the first one with laminar viscosity, thermal conductivity and diffusion coefficients.
Then we adapted the main routines of the **CSolver** class in order to adequately call the functions implemented in the other classes for computing residuals and imposing boundary conditions.
Moreover we added all the options for reading extra informations typical of multispecies flows and needed by the configuration file such as free-stream or inlet mass fractions.
Eventually we chose to rely on an external library to read chemical reactions and its parameters (Arrhenius coefficents, activation temprature, ...) and to compute all the physical-chemical properties required.

In the following subsections as summary we will briefly focus on some relevant aspects of the classes related to **CSolver**, **CNumerics** and **Cvaribale** and on the library.

4.2.1 CSolver

In this class the solution procedure is defined and each child represent a solver for a particular set of governing equations: ours will be solved through class **CReactiveEulerSolver** or **CReactiveNSSolver** depending on whether Euler or Navier-Stokes equations have to be investigated.

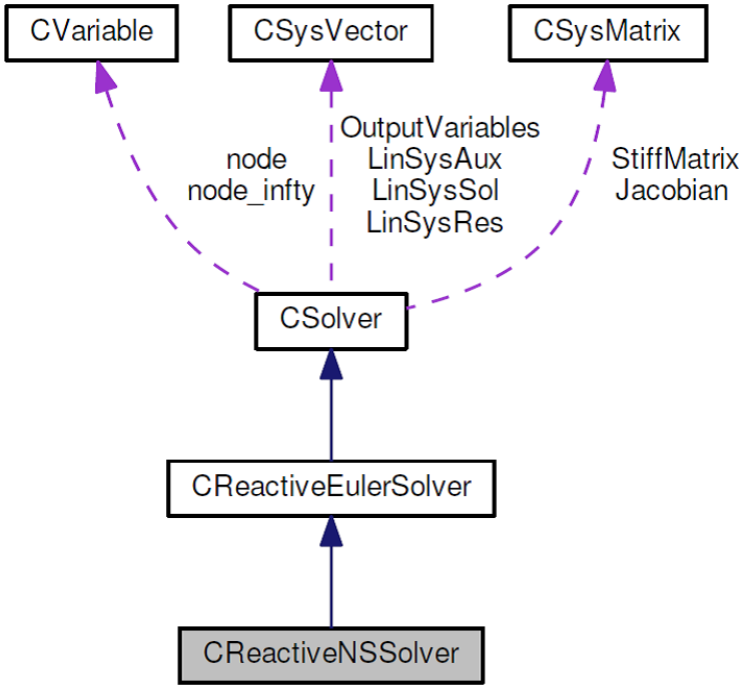


Figure 4.2: Diagram for solver class

These solver classes contain functions for computing each spatial term of the PDE: we find loops over the mesh edges to compute convective and viscous fluxes and loops over the mesh nodes to compute source terms as well as routines for imposing suitable boundary conditions.

We report briefly the secant method to find temperature in case of subsonic inlet which has been discussed in the previous section

```
/*--- Auxiliary function to impose adiabaticity ---*/
auto f = std::function<su2double(su2double)>([&](su2double T){
  su2double hb = library->ComputeEnthalpy(T, Ys);
  su2double cb = std::sqrt(Gamma*Rgas*T);
  su2double Vb = (Riemann - 2.0*cb/Gamma_Minus_One)/alpha;
  return hb + 0.5*Vb*Vb;
});

/*--- Set parameters for secant method to find temperature ---*/
bool NRconvg, Bconvg;
su2double NRtol = 1.0e-9; // Tolerance for the Secant method
su2double Btol = 1.0e-6; // Tolerance for the Bisection method
unsigned short maxNIter = 15;     // Maximum Secant method iterations
unsigned short maxBIter = 100;     // Maximum Bisection method iterations
unsigned short iIter;

su2double Told = Ttot + 1.0;
```

```
su2double Tcurr = Ttot;
su2double Tnew;
NRconvg = false;

/*--- Execute a secant root-finding method to find the inlet temperature
    (TRAPEZOIDAL) ---*/
for(iIter = 0; iIter < maxNIter; ++iIter) {
  su2double tmp = f(Tcurr);
  su2double F = tmp - Tot_Enthalpy;
  su2double dF = tmp - f(Told);
  Tnew = Tcurr - F*(Tcurr - Told)/dF;

  /*--- Check for convergence ---*/
  if(std::abs(Tnew - Tcurr) < NRtol) {
    NRconvg = true;
    break;
  }
  else {
    Told  = Tcurr;
    Tcurr = Tnew;
  }
}

if(NRconvg)
  V_inlet[T_INDEX_PRIM] = Tcurr;
else {
  /*--- Execute the bisection root-finding method ---*/
  Bconvg = false;
  su2double Ta = config->GetTemperatureMin()/config->GetTemperature_Ref();
  su2double Tb = Ttot;
  for(iIter = 0; iIter < maxBIter; ++iIter) {
    Tcurr = (Ta + Tb)/2.0;
    su2double F = f(Tcurr) - Tot_Enthalpy;

    if(std::abs(F) < Btol) {
      V_inlet[T_INDEX_PRIM] = Tcurr;
      Bconvg = true;
      break;
    }
    else {
      if(F > 0.0)
        Ta = Tcurr;
      else
        Tb = Tcurr;
    }
  }

  /*--- If absolutely no convergence, then something is going really wrong
      ---*/
  if(!Bconvg)
    throw std::runtime_error("Convergence not achieved for bisection method
        in inlet boundary condition");
}
```

Moreover we report a typical loop used in order to call the functions that evaluate the

flux and that are implemented in the **CNumerics** class as explained later on.

```
/*--- Loop over all the edges ---*/
for(iEdge = 0; iEdge < geometry->GetnEdge(); ++iEdge) {
  /*--- Points in edge and normal vectors ---*/
  iPoint = geometry->edge[iEdge]->GetNode(0);
  jPoint = geometry->edge[iEdge]->GetNode(1);
  numerics->SetNormal(geometry->edge[iEdge]->GetNormal());

  /*--- Get primitive variables ---*/
  auto V_i = node[iPoint]->GetPrimitive();
  auto V_j = node[jPoint]->GetPrimitive();

  ...

  /*--- Set primitive variables without reconstruction ---*/
  numerics->SetPrimitive(V_i, V_j);
  if(implicit)
    numerics->SetSecondary(node[iPoint]->GetdPdU(),node[jPoint]->GetdPdU());

  /*--- Compute the residual ---*/
  numerics->ComputeResidual(Res_Conv, Jacobian_i, Jacobian_j, config);

  /*--- Check for NaNs before applying the residual to the linear system
      ---*/
  bool err = !std::none_of(Res_Conv, Res_Conv + nVar,
      [](su2double elem){return std::isnan(elem);});
  if(implicit) {
   if(!err) {
     for(iVar = 0; iVar < nVar; ++iVar) {
       err = !std::none_of(Jacobian_i[iVar], Jacobian_i[iVar] + nVar,
           [](su2double elem){return std::isnan(elem);});
       err = err || !std::none_of(Jacobian_j[iVar], Jacobian_j[iVar] + nVar,
           [](su2double elem){return std::isnan(elem);});
       if(err)
         break;
     }
   }
  }

  /*--- Update residual value ---*/
  if(!err) {
   LinSysRes.AddBlock(iPoint, Res_Conv);
   LinSysRes.SubtractBlock(jPoint, Res_Conv);

   /*--- Set implicit Jacobians ---*/
   if(implicit) {
     Jacobian.AddBlock(iPoint, iPoint, Jacobian_i);
     Jacobian.AddBlock(iPoint, jPoint, Jacobian_j);
     Jacobian.SubtractBlock(jPoint, iPoint, Jacobian_i);
     Jacobian.SubtractBlock(jPoint, jPoint, Jacobian_j);
    }
  }
  else
    throw std::runtime_error("NaN found in the upwind residual");
} /*--- End loop over edges ---*/
```

4.2.2 CVariable

As we can see in Figure (4.2) the solver classes refer to the CVariable class for storing unknowns and other variables pertinent to the PDE at each mesh node and for our simulations we created, as stated before, two suitable child classes called **CReactiveEulerVariable** and **CReactiveNSVariable** depending on the type of problem we need to investigate.

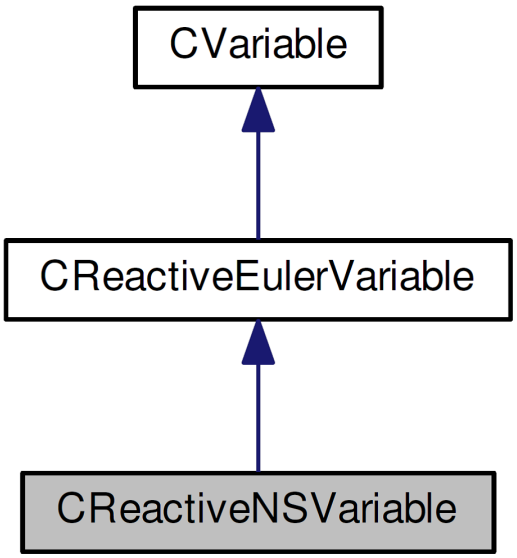


Figure 4.3: Diagram for variable class

An important consideration when we deal with equations or system of equations like (2.17) is the non-dimensionalization in order to avoid undesirable scaling effects due to units. The particular scheme chosen for SU2 can be found in Table (4.1) and (4.2)

Variables	Value	SI
Length	l_{ref}(input)	m
Pressure	p_{ref}(input)	$\mathrm{kg\,m^{-1}\,s^{-2}}$
Density	ρ_{ref}(input)	$\mathrm{kg\,m^{-3}}$
Temperature	T_{ref}(input)	K

Table 4.1: Independent variables for which reference values can be arbitrarily chosen

Variables	Value	SI
Velocity	$u_{ref} = \sqrt{p_{ref}/\rho_{ref}}$	$\mathrm{m\,s^{-1}}$
Time	$t_{ref} = l_{ref} u_{ref}$	s
Dynamic Viscosity	$\mu_{ref} = \rho_{ref} u_{ref} l_{ref}$(input)	$\mathrm{kg\,m^{-1}\,s^{-1}}$
Specific energy	$e_{ref} = u_{ref}^2$	$\mathrm{m^2\,s^{-2}}$
Specific enthalpy	$h_{ref} = e_{ref}$	$\mathrm{m^2\,s^{-2}}$
Gas Constant	$R_{ref} = e_{ref}/T_{ref}^2$	$\mathrm{m^2\,s^{-2}\,K^{-1}}$
Heat capacity (constant pressure)	$c_{p_{ref}} = R_{ref}$	$\mathrm{m^2\,s^{-2}\,K^{-1}}$
Heat capacity (constant volume)	$c_{v_{ref}} = R_{ref}$	$\mathrm{m^2\,s^{-2}\,K^{-1}}$
Turbulent kinetic energy	$k_{ref} = u_{ref}^2$	$\mathrm{m^2\,s^{-2}}$
Turbulent specific dissipation	$\omega_{ref} = u_{ref}/l_{ref}$	$\mathrm{s^{-1}}$

Table 4.2: Reference values for all other variables

Unlike the original implementation we allow the user to choose the reference length which was prior fixed to 1.0 m in order to allow more flexibility and to take into account the mesh dimension which will be hopefully related to the characteristic length of the problem.

```
//Length_Ref = 1.0; //<---- NOTE: this should be given an option or set as
    a const
/*!\brief REF_LENGTH\n DESCRIPTION: Reference length for
    adimensionalitazion (1.0 m by default) \ingroup Config*/
addDoubleOption("REF_LENGTH", Length_Ref, 1.0);
```

Moreover we added some suitable variables in order to access a certain quantity inside the conserved or primitive array: in this way another developer can simply modify these values if a different order is needed.

```
/**
* Mapping between the primitive variable name and its position in the
    physical data
*/
static constexpr unsigned short T_INDEX_PRIM = 0;
static constexpr unsigned short VX_INDEX_PRIM = 1;
static unsigned short P_INDEX_PRIM;
static unsigned short RHO_INDEX_PRIM;
static unsigned short H_INDEX_PRIM;
static unsigned short A_INDEX_PRIM;
static unsigned short RHOS_INDEX_PRIM;

/**
* Mapping between the solution variable name and its position in the
    physical data
*/
static constexpr unsigned short RHO_INDEX_SOL = 0;
static constexpr unsigned short RHOVX_INDEX_SOL = 1;
static unsigned short RHOE_INDEX_SOL;
static unsigned short RHOS_INDEX_SOL;
```

The most important function is the one that allows at the beginning of each iteration to pass from conserved to primitive variables before computing fluxes: it is called **Cons2PrimVar** and its interesting part is the computation of temperature from total energy

```
/*--- Translational-Rotational Temperature ---*/
const su2double Rgas =
    library->ComputeRgas(Ys)/config->GetGas_Constant_Ref();
const su2double C1 = (-rhoE + 0.5*rho*sqvel)/(rho*Rgas);
const su2double C2 = 1.0/Rgas;

/*--- Pick initial state and start algorithm ---*/
T = V[T_INDEX_PRIM];
Told = T + 1.0;
for(iIter = 0; iIter < maxNIter; ++iIter) {
  /*--- Execute a secant root-finding method to find T ---*/
  su2double dim_temp, dim_temp_old;
  dim_temp = T*config->GetTemperature_Ref();
  dim_temp_old = Told*config->GetTemperature_Ref();
  if(US_System) {
   dim_temp *= 5.0/9.0;
   dim_temp_old *= 5.0/9.0;
  }
  hs_old = library->ComputeEnthalpy(dim_temp_old,
      Ys)/config->GetEnergy_Ref();
  hs = library->ComputeEnthalpy(dim_temp, Ys)/config->GetEnergy_Ref();
  if(US_System) {
   hs_old *= 3.28084*3.28084;
   hs *= 3.28084*3.28084;
  }

  f = T - C1 - C2*hs;
  df = T - Told + C2*(hs_old-hs);
  Tnew = T - f*(T-Told)/df;

  /*--- Check for convergence ---*/
  if(std::abs(Tnew - T) < NRtol) {
    NRconvg = true;
    break;
  }
  else {
    Told = T;
    T = Tnew;
  }
}

/*--- If the secant method has converged, assign the value of T.
Otherwise execute a bisection root-finding method ---*/
if(NRconvg)
  V[T_INDEX_PRIM] = T;
else {
  Bconvg = false;
  su2double Ta = Tmin;
  su2double Tb = Tmax;
  for(iIter = 0; iIter < maxBIter; ++iIter) {
    T = (Ta + Tb)/2.0;
    su2double dim_temp = T*config->GetTemperature_Ref();;
```

```
    if(US_System)
      dim_temp *= 5.0/9.0;
    hs = library->ComputeEnthalpy(dim_temp, Ys)/config->GetEnergy_Ref();
    if(US_System)
      hs *= 3.28084*3.28084;
    f = T - C1 - C2*hs;

    if(std::abs(f) < Btol) {
      V[T_INDEX_PRIM] = T;
      Bconvg = true;
      break;
    }
    else {
      if(f > 0)
        Ta = T;
      else
        Tb = T;
    }
  }

  /*--- If absolutely no convergence, then something is going really wrong
      ---*/
  if(!Bconvg)
    throw std::runtime_error("Convergence not achieved for bisection
        method");
}
```

4.2.3 CNumerics

This class discretizes each system of governing equations using the numerical schemes specified in the input file. There are several child that provide discretization techniques for convective fluxes,viscous fluxes and source terms. During a single iteration methods in the CNumerics classes would compute the flux contributions and Jacobians (in case of implicit computations) at each node (using the variables stored in the CVariable class). These flux and Jacobian values are transferred back to the CSolver class which calls routines within CSysMatrix in order to solve the resulting linear system of equations for the solution update.

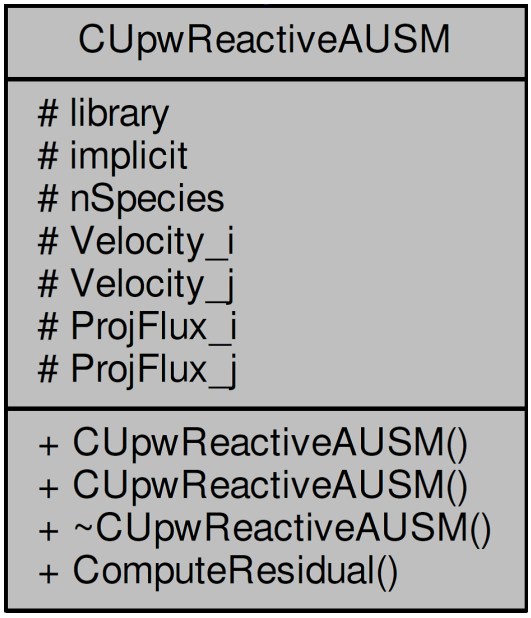


Figure 4.4: UML diagram for the class CReactiveUpwAUSM

Herein we reported the UML diagram with some members of the new class **CUpwReactiveAUSM** used for the convective fluxes: the function that effectively computes the flux and eventually the Jacobians is called **ComputeResidual**.
It is worth to notice that the SU2 Code widely employes **run-time polymorphism**: therefore in order to implement any kind of new physical model it is necessary to derive from the base class **CNumerics** and to implement accordingly the **virtual** function **ComputeResidual**. For our purpose three more classes are needed: one for viscous fluxes at boundary nodes (**CAvgGradReactive_Boundary**) where we cannot apply the corrected average gradient formula (3.15), one for viscous fluxes at internal nodes(**CAvgGradReactive_Flow**) and one for chemistry source terms. (**CSourceReactive**).

4.2.4 Reacting Model Library

This section describes the structure of the library implemented for computing the physical and chemical properties in the mixture.
In order to allow each user to use its own version of the library and to handle the creation of polymorphic object we use a simple version of the **factory** design pattern.

```
/*!
* \class Factory
* \brief Class for loading libraries at run-time.
* \author G. Orlando
*/
template<class Base>
class Factory: public Common::NotCopyable<Factory<Base>> {
public:

/*
* \brief Constructor of this simple factory
* \param[in] lib_name - Name of the desired library
* \param[in] config_name - Name of the file to read in order to configure
    the library
* \param[in] lib_path - Path where the library is present
*/
Factory(const std::string& lib_name, const std::string& config_name, const
    std::string& lib_path);

/*
* \brief Factory destrcutor
*/
~Factory() {}

/*
* \brief Get the library
*/
std::shared_ptr<Base> GetLibraryPtr(void) const {
  return my_library;
}

private:
  std::shared_ptr<Base> my_library; /*!< \brief Pointer to Base in order to
      access concrete version. */

}; /*--- End of class Factory ---*/
```

Class **Factory** is templatized with the polymorphic *Base* type as a parameter and serves as a registry point for all the providers of that type.

In our case the *Base* type is represented by the class **PhysicalChemicalLibrary**, while **ReactingModelLibrary** is the concrete implementation of the functions virtually declared in the *Base*.

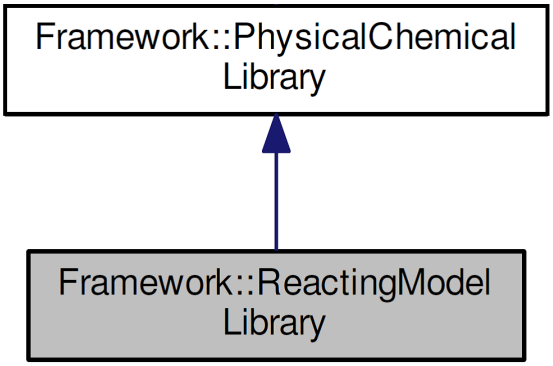


Figure 4.5: Hierarchy for reacting model library

In the solver classes we add as a member a static shared pointer to the *Base* class so that we have one version of the library and all functions can rely on that if necessary

```
class CReactiveEulerVariable: public CVariable {
public:
  typedef std::vector<su2double> RealVec;
  typedef su2double** SU2Matrix;
  typedef std::shared_ptr<Framework::PhysicalChemicalLibrary> LibraryPtr;
....
}
/*! \class CReactiveEulerSolver
*  \brief Main class for defining a solver for chemically reacting inviscid
    flows.
*  \author G. Orlando.
*/
class CReactiveEulerSolver: public CSolver {
public:
  using RealVec = CReactiveEulerVariable::RealVec;
  using RealMatrix = CReactiveNSVariable::RealMatrix;
  using LibraryPtr = CReactiveEulerVariable::LibraryPtr;

protected:
  static LibraryPtr library; /*!< \brief Smart pointer to the library that
      computes physical-chemical properties. */
....
}
```

The reading of mixture data, chemical reactions, thermodynamic and transport properties data is delegated to the library through a suitable function called **Setup**

```
/*--- Setup library ---*/
void ReactingModelLibrary::Setup(void) {
if(!Lib_Setup) {
  Le = 1.0;
  /*--- If nobody has configured the library path, we try to do it here
      with a default value ---*/
  if(Lib_Path == "") {
   std::cout<<"Library path set to default"<<std::endl;
   auto base_dir = std::experimental::filesystem::current_path().string();
   Lib_Path = base_dir;
  }
```

```
std::vector<std::string> list_file;
std::ifstream config_file(Config_File);
if(config_file.is_open()) {
  while(config_file.good() && !config_file.eof()) {
   std::string curr_line;
   std::getline(config_file,curr_line);
   if(!curr_line.empty() && !std::ispunct(curr_line.at(0)))
     list_file.push_back(curr_line);
  }
}
else {
  std::cerr<<"Unable to open the specified file with all the file names
      for setting library."<<std::endl;
  std::exit(1);
}

/*--- Read mixture file: it needs to be the first to check exactness of
    chemical reactions and properties ---*/
std::string file_mix = list_file.at(0);
ReadDataMixture(file_mix);
std::cout<<"Mixture Data read"<<std::endl;

/*--- Check we have the right number of files ---*/
using size_type = std::vector<std::string>::size_type;
size_type max_n_file = 2*nSpecies + 2;
size_type n_file = list_file.size();
SU2_Assert((n_file == max_n_file) || (n_file == max_n_file - 1), "The
    number of files present in the configuration file is wrong");

/*--- Set the specific gas constants ---*/
SetRiGas();

/*--- Read chemistry file (if present) ---*/
int buffer_chemistry = 1;
nReactions = 0;
if(n_file == max_n_file) {
  /*--- We assume that chemistry file is the second in the list if its
      present ---*/
  std::string file_chem = list_file[1];
  /*--- Herein we read chemical reaction with a suitable parser.
  Inline the reactions we specify the coefficients for the exponents to
      compute rates ---*/
  ReadDataChem(file_chem);
  std::cout<<"Chemical Reactions read"<<std::endl;
  buffer_chemistry = 0;
}

/*--- We assume that data correspond to the species declared at the
    beginning of the file
      and a transport file is followed by a thermodynamic file
      (I can't check the content so it seems reasonable) ---*/
std::string file_transp, file_thermo;
for(unsigned short iSpecies = 0; iSpecies < nSpecies; ++iSpecies) {
 file_transp = list_file[iSpecies*2 + 2 - buffer_chemistry];
 ReadDataTransp(file_transp);
 file_thermo = list_file[iSpecies*2 + 3 - buffer_chemistry];
```

```
   ReadDataThermo(file_thermo);
  }
  Lib_Setup = true;
  std::cout<<"Library set."<<std::endl;
  std::cout<<std::endl;
}
else
  throw Common::NotSetup("Trying to setup again without calling unsetup
      first.");
}
```

where **Common::NotSetup** is a simple **exception** to underline that the library has not been correctly instantiated.

We report here also the function **ReadDataChem** employed to read all parameters related to chemical reactions

```
/*--- Reading data about chemistry. ---*/
void ReactingModelLibrary::ReadDataChem(const std::string& f_name) {
  /*--- Local variables ---*/
  std::string line;
  unsigned n_line = 0;
  unsigned n_reac_read = 0;

  std::ifstream chemfile(Lib_Path + "/" + f_name);
  if(chemfile.is_open()) {
    /*--- Clear for safety ---*/
    Stoich_Coeffs_Reactants.resize(0,0);
    Stoich_Coeffs_Reactants.resize(0,0);
    Stoich_Coeffs_Products_Exp.resize(0,0);
    Stoich_Coeffs_Reactants_Exp.resize(0,0);
    Reversible_Reactions.clear();
    As.clear();
    Betas.clear();
    Temps_Activation.clear();
    while(chemfile.good() && !chemfile.eof()) {
      std::getline(chemfile,line);

      /*--- Check if we encounter the termination character ---*/
      if(line == "STOP")
        break;
      /*--- We avoid clearly reading empty lines and comments in the file
          ---*/
      if(!line.empty() && !std::ispunct(line.at(0))) {
        if(n_line == 0) {
          std::istringstream curr_line(line);
          curr_line>>nReactions;
          SU2_Assert(!curr_line.fail(), "You have to specify the number of
              reactions before proceding");

         /*--- Resize and reserve space for vectors ---*/
         Stoich_Coeffs_Reactants.resize(nSpecies,nReactions);
         Stoich_Coeffs_Reactants.setZero();
         Stoich_Coeffs_Products.resize(nSpecies,nReactions);
         Stoich_Coeffs_Products.setZero();
```

```
      Stoich_Coeffs_Reactants_Exp.resize(nReactions,nSpecies);
      Stoich_Coeffs_Reactants_Exp.setZero();
      Stoich_Coeffs_Products_Exp.resize(nReactions,nSpecies);
      Stoich_Coeffs_Products_Exp.setZero();

      Forward_Rates.resize(nReactions);
      Backward_Rates.resize(nReactions);
      Kc.resize(nReactions);
      Kc_Derivatives.resize(nReactions);

      Reversible_Reactions.reserve(nReactions);
      As.reserve(nReactions);
      Betas.reserve(nReactions);
      Temps_Activation.reserve(nReactions);

      n_line++;
     }
     else if(n_line == 1) {
     std::istringstream curr_line(line);
     std::string kind_units;
     curr_line>>kind_units;
     if(kind_units == "CGS")
       CGS_Units = true;
     else if(kind_units == "SI")
       CGS_Units = false;
     else
       throw std::out_of_range("Unknown option for the type of units
           measure");

     n_line++;
    }
     else {
     bool is_rev;
       if(n_line 'mod' 2 == 0 && n_line < nReactions + 3) {
         is_rev = (line.find('<') != std::string::npos);
         Reversible_Reactions.push_back(is_rev);
         n_reac_read++;
         ReadReactSpecies(line,is_rev,n_reac_read);
       }
       else if(n_line 'mod' 2 == 1 && n_line < nReactions + 4) {
       ReadChemCoefs(line);
    }
    else {
      ReadExtraData_Rates(line);
      ReadExtraData_ForwardExponent(line);
      ReadExtraData_BackwardExponent(line);
    }

     n_line++;
    }
  }
}
SU2_Assert(n_reac_read == nReactions, "The number of reactions detected
    doesn't match nReactions");
chemfile.close();
unsigned short iReac, iSpecies;
```

```
  /*--- Try automatic computations of exponents of products in case
      backward data were not already available ---*/
  for(iReac = 0; iReac < nReactions; ++iReac) {
    if(Reversible_Reactions[iReac] &&
        Available_Backward_Rate.count(iReac) == 0) {
    for(iSpecies = 0; iSpecies < nSpecies; ++iSpecies)
      Stoich_Coeffs_Products_Exp(iReac,iSpecies)=
          Stoich_Coeffs_Reactants_Exp(iReac,iSpecies) +
      Stoich_Coeffs_Products(iSpecies,iReac) -
          Stoich_Coeffs_Reactants(iSpecies,iReac);
    }
  }

  /*--- Update to SI units the Arrhenius constants if needed ---*/
  if(CGS_Units) {
    for(iReac = 0; iReac < nReactions; ++ iReac) {
      double sum_forward_exp = Stoich_Coeffs_Reactants_Exp.row(iReac).sum();
      As[iReac] *= std::pow(10.0, 6.0*(1.0 - sum_forward_exp));
      if(Available_Backward_Rate.count(iReac) == 1) {
        double sum_backward_exp =
            Stoich_Coeffs_Products_Exp.row(iReac).sum();
        As_back[iReac] *= std::pow(10.0, 6.0*(1.0 - sum_backward_exp));
      }
    }
  }

  /*--- Save species with negative exponents for forward rates---*/
  Reactant_Species_Negative_Exponent.resize(nReactions);
  for(iReac = 0; iReac < nReactions; ++iReac) {
    for(iSpecies = 0; iSpecies < nSpecies; ++iSpecies) {
      if(Stoich_Coeffs_Reactants_Exp(iReac,iSpecies) < 0.0)
        Reactant_Species_Negative_Exponent[iReac].push_back(iSpecies);
    }
  }

  /*--- Save species with negative exponents for backward rates---*/
  Product_Species_Negative_Exponent.resize(nReactions);
  for(iReac = 0; iReac < nReactions; ++iReac) {
    for(iSpecies = 0; iSpecies < nSpecies; ++iSpecies) {
      if(Stoich_Coeffs_Products_Exp(iReac,iSpecies) < 0.0)
        Product_Species_Negative_Exponent[iReac].push_back(iSpecies);
    }
  }

 }
 else {
   std::cerr<<"Unable to open the chemical file: "<<f_name<<std::endl;
   std::exit(1);
 }
}
```

As it can be easily noticed we allow the user to express the Arrhenius constants both in **CGS** units (mol/cm^3) and **SI** units (mol/m^3) and the same holds also for the activation temperature where we can express the activation energy in cal/mol or directly the temperature in K.

Moreover we save the species with negative exponents in order to avoid the rates going to infinity in case of zero or vanishing mass fractions.

Let us discuss now the models employed in the library for computing **transport** (viscosity and conductivity) and **thermodynamic** (specific heat, enthalpy and entropy) properties as well as **diffusion** coefficients.

Viscosity

We need adequate relations for the molecular viscosity of single species μ_i and for the one of the mixture μ; regarding molecular viscosity of the single species, the interpolations from [3] is used for all the species except for 1,3-butadiene (C_4H_6), which is a species that will be involved in one of our simulations and is not available in given reference. It is an interpolation of the form:

$$\ln \mu_i = A_i \ln T + \frac{B_i}{T} + \frac{C_i}{T^2} + D_i \tag{4.1}$$

where coefficients A_i, B_i, C_i and D_i are given in [3] for $200 - 1000$ K and $1000 - 6000$ K temperature ranges. In addition we remark that data ranges for H_2O and O are available respectively from 300 K and from 1000 K and below these values the viscosity of these species is considered constant and equal to the first available datum.

For C_4H_6 the Chapman-Enskog correlation from [10] is used:

$$\mu_{C_4H_6} = 26.69 \frac{\sqrt{MT}}{\sigma^2 \Omega_v}$$

with

$$\Omega_v = a(T^*)^{-b} + c\exp(-dT^*) + e\exp(-fT^*)$$
$$\sigma = 0.809 V_c^{\frac{1}{3}}$$
$$T^* = T\frac{k_b}{\epsilon_c}$$

where k_b is the Boltzmann constant, ϵ_c is the characteristic energy and V_c is the critical volume. Moreover:

$$a = 1.16145 \qquad d = 0.77320$$
$$b = 0.14874 \qquad e = 2.16178$$
$$c = 0.52487 \qquad f = 2.43787$$

Results from the expression for viscosity of 1,3 butadiene considered so far anyway do not take into account molecular polarity. Therefore the following relation from [11] is used:

$$\mu_{C_4H_6} = 40.785 \frac{F_c \sqrt{MT}}{V_c^{\frac{2}{3}} \Omega_v}$$

where F_c is the correction factor whose expression depends on the polarity characteristics of C_4H_6 and has the form

$$F_c = 1 - 0.2765AF$$

where AF is the acentric factor, a measure of the "non sphericity" of the 1,3 butadiene molecule.

All values of required parameters are reported in Table (4.3).

Moreover data for H_2O and for monoatomic gases such as O and H are available respectively from 300 K and 1000 K: below these values, viscosity was not extrapolated but considered constant and with value corresponding to the first available datum.

For the molecular viscosity of the mixture μ the well known Wilke's formula from [20] is used:

$$\mu = \sum_{i=1}^{N_S} \frac{\mu_i X_i}{\sum_{j=1}^{N_S} X_j \phi_{ij}} \tag{4.2}$$

where

$$\phi_{ij} = \frac{\left[1 + \left(\frac{\mu_i}{\mu_j}\right)^{\frac{1}{2}} \left(\frac{M_j}{M_i}\right)^{\frac{1}{4}}\right]^2}{\sqrt{8}\left[1 + \left(\frac{M_i}{M_j}\right)\right]^{\frac{1}{2}}}$$

Let us analyse more in detail the relation (4.2) in order to get a form that will be useful later on for a comparison with the formula used for thermal conductivity: first of all let us notice that $\phi_{ii} = 1$ so that

$$\sum_{i=1}^{N_S} \frac{\mu_i X_i}{\sum_{j=1}^{N_S} X_j \phi_{ij}} = \sum_{i=1}^{N_S} \frac{\mu_i X_i}{X_i + \sum\limits_{\substack{j=1 \\ j\neq i}}^{N_S} X_j \phi_{ij}} = \sum_{i=1}^{N_S} \frac{\mu_i}{1 + \frac{1}{X_i} \sum\limits_{\substack{j=1 \\ j\neq i}}^{N_S} X_j \phi_{ij}} = \sum_{i=1}^{N_S} \frac{\mu_i}{1 + \frac{1}{X_i}\Sigma_\phi} \tag{4.3}$$

where

$$\Sigma_\phi = \sum_{\substack{j=1 \\ j\neq i}}^{N_S} X_j \phi_{ij}$$

Thermal Conductivity

For thermal conductivity of the species κ_i, the same kind of interpolation as the one previously shown for viscosity from [3] can be used. Coefficients A_i, B_i, C_i and D_i coefficients are conductivity-specific and are given by the reference.
This again does not apply to 1,3-butadiene, which is not available in given reference, as for viscosity. For C_4H_6 different approaches have been compared by [1]: two modified Eucken model (from [21] and [22]) and Chung method (from [11]) all summarized in [21]. All these models require ideal gas hypothesis and are expressed as follows:

$$\kappa_{C4H6} = \frac{\mu_{C4H6} C_v}{M_{C4H6}} \left(1.30 + 1.7614 \frac{R_u}{C_v}\right) \tag{4.4}$$

Here C_v is the specific heat at constant volume computed as $C_v = C_p - R_u$ according the law of ideal gases and M_{C4H6} is the molar mass of 1,3-butadiene. Moreover we remark that the dynamic viscosity μ_{C4H6} has been calculated in the previous section while the computation of specific heat at constant pressure, which for an ideal gas depends only on temperature, will be explored in the next section.

Stiel et al. in [22] proposed a modification of (4.4) which reads:

$$\kappa_{C4H6} = \frac{\mu_{C4H6} C_v}{M_{C4H6}} \left(1.15 + 2.033 \frac{R_u}{C_v}\right) \tag{4.5}$$

Chung et al. in [11] proposed a method that expresses the thermal conductivity as:

$$\kappa_{C4H6} = \frac{\mu_{C4H6} C_v}{M_{C4H6}} 3.77 R_u \Psi \tag{4.6}$$

which Ψ is a function of three parameters α, β and Z of the form:

$$\Psi = 1 + \alpha \left[\frac{0.215 + 0.28288\alpha - 1.061\beta + 0.26665Z}{0.6366 + \beta Z + 1.061\alpha\beta} \right]$$

with:

$$\alpha = \frac{C_v}{R_u} - \frac{3}{2} = \frac{C_p}{R_u} - \frac{5}{2}$$

$$\beta = 0.7682 - 0.7109AF + 1.3168(AF)^2$$

$$Z = 2.0 + 10.5 \left(\frac{T}{T_c} \right)^2$$

Let us summarize all data needed for transport properties of 1,3-butadiene For the

M[g mol^{-1}]	54.092
V_c[cm^3 mol^{-1}]	220
ϵ_c [J]	$4.6614021 \cdot 10^{-21}$
AF	0.192
T_c[K]	425.17

Table 4.3: Parameters C_4H_6

thermal conductivity of the mixture κ a Wilke's formula analogous to (4.3) holds, but an approach from [23] is chosen. This method is valid for polyatomic gas mixtures but represents an adequate approximation even if monoatomic species are present and reads:

$$\kappa = \sum_{i=1}^{N_S} \frac{\kappa_i}{1 + \frac{1}{X_i} \Sigma_\psi} \tag{4.7}$$

where

$$\Sigma_\psi = \sum_{\substack{j=1 \\ j \neq i}}^{N_S} X_j \psi_{ij}$$

$$\psi_{ij} = \frac{1.065 \left[1 + \left(\frac{\kappa_i}{\kappa_j} \right)^{\frac{1}{2}} \left(\frac{M_j}{M_i} \right)^{\frac{1}{4}} \right]^2}{\sqrt{8} \left[1 + \left(\frac{M_i}{M_j} \right) \right]^{\frac{1}{2}}}$$

Entropy, Enthalpy and Specific Heat

Single species entropy s_i, static enthalpy h_i and specific heat at constant pressure $C_{p,i}$ are taken from NASA polynomials as given by [4]. The polynomials have the form:

$$\frac{s_\circ}{R_u} = a_{1i} \ln T + a_{2i} T + \frac{a_{3i} T^2}{2} + \frac{a_{4i} T^3}{3} + \frac{a_{5i} T^4}{4} + a_{7i}$$

$$\frac{h_i}{R_u T} = a_{1i} + \frac{a_{2i} T}{2} + \frac{a_{3i} T^2}{3} + \frac{a_{4i} T^3}{4} + \frac{a_{5i} T^4}{5} + \frac{a_{6i}}{T}$$

$$\frac{c_{p,i}}{R_u} = a_{1i} + a_{2i} T + a_{3i} T^2 + a_{4i} T^3 + a_{5i} T^4$$

It is important to notice that the static enthalpy given by the polynomial expression includes the enthalpy of formation at $T_{ref} = 298\text{K}$.

Finally in order to obtain the mixture entropy s, the mixture static enthalpy h and the mixture specific heat at constant pressure C_p we average over mass fractions, i.e.:

$$s^\circ = \sum_{i=1}^{N_S} Y_i s_i^\circ \tag{4.8}$$

$$h = \sum_{i=1}^{N_S} Y_i h_i \tag{4.9}$$

$$C_p = \sum_{i=1}^{N_S} Y_i C_{p,i} \tag{4.10}$$

In this work we do not apply directly the aforementioned polynomials to compute transport and thermodynamic properties, but we employ spline interpolation techniques (see Appendix (A.8)) in order to obtain flexibility in the use of several user-defined models and to gather the computational way to proceed; this is performed in the following manner: in a pre-processing stage data tables for thermodynamic/transport properties are generated as text files using literature polynomial data with user-defined temperature interval ΔT and then the spline interpolation coefficients are generated during the setup phase.
Eventually in order to determine the interval which a desired temperature belongs to, the integer search algorithm has been employed: in this way the extrema of interval can be found through a simple division with a noticeable speed-up in the algorithm.

Binary diffusion coefficients

For what concerns diffusion fluxes in order to solve (2.21) we need an expression for binary diffusion coefficients. As reported in [5] their expression is the following

$$D_{ij} = \frac{7.1613 \cdot 10^{-25}}{\rho} \frac{M \left[T \left(\frac{1}{M_i} + \frac{1}{M_j} \right) \right]^{1/2}}{\Omega_{ij}}$$

which requires the computation of collision integrals Ω_{ij}: this is very expansive from a computational point of view and so we decide to exploit a semi-empirical formula from [12]

$$D_{ij} = \frac{10^{-3} T^{1.75}}{p M_{ij}^{\frac{1}{2}} \left[(\Sigma_v)_i^{\frac{1}{3}} + (\Sigma_v)_j^{\frac{1}{3}} \right]^2} \tag{4.11}$$

where

$$M_{ij} = \left[\frac{1}{M_i} + \frac{1}{M_j} \right]^{-1}$$

and $(\Sigma_v)_i$ is the molecular diffusion volume of species i as the sum of atomic and structural volume.

It is important to remark that in (4.11) the temperature must be expressed in K and the pressure in atmospheres in order to obtain the result in $\left[\text{cm}^2\,\text{s}^{-1}\right]$ using data in Table (4.4).

Species	Diffusion Volumes $[cm^3/mol]$
CO	18.00
CO_2	26.90
C_4H_6	77.46
H	2.31
H_2	6.12
H_2O	13.10
O	6.11
O_2	16.30
OH	8.42
N_2	17.90

Table 4.4: Molecular diffusion volumes of considered species (from [1])

4.3 Execution

The main program for solving fluid dynamics applications calls essentially three routines as reported:

```
/*--- Multi-zone problem: instantiate the multi-zone driver class by default
or a specialized driver class for a particular multi-physics problem. ---*/

driver = new CFluidDriver(config_file_name, nZone, nDim, MPICommunicator);

delete config;
config = NULL;

/*--- Launch the main external loop of the solver ---*/

driver->StartSolver();

/*--- Postprocess all the containers, close history file, exit SU2 ---*/

driver->Postprocessing();
```

Inside the constructor of the class **CFluidDriver** we call several functions including `Iteration_Preprocessing`, `Solver_Preprocessing`, `Integration_Preprocessing` and `Numerics_Preprocessing`.

`Iteration_Preprocessing` allocates the class **MeanFlowIteration** as shown here:

```
case REACTIVE_EULER: case REACTIVE_NAVIER_STOKES:
if (rank == MASTER_NODE)
 std::cout << ": Euler/Navier-Stokes/RANS fluid iteration." << std::endl;
iteration_container[iZone] = new
    CMeanFlowIteration(config_container[iZone]);
break;
```

`Solver_Preprocessing` allocates the appropriate class **CSolver** and calls the `Preprocessing` function inside the **CSolver** class.

```
if(reactive_euler) {
 if(compressible) {
  solver_container[iMGlevel][FLOW_SOL] = new
      CReactiveEulerSolver(geometry[iMGlevel], config, iMGlevel);
  solver_container[iMGlevel][FLOW_SOL]->
  Preprocessing(geometry[iMGlevel], solver_container[iMGlevel], config,
              iMGlevel, NO_RK_ITER, RUNTIME_REACTIVE_SYS, false);
 }
 }
 if(reactive_ns) {
  if(compressible) {
   solver_container[iMGlevel][FLOW_SOL] = new
       CReactiveNSSolver(geometry[iMGlevel], config, iMGlevel);
   solver_container[iMGlevel][FLOW_SOL]->
   Preprocessing(geometry[iMGlevel], solver_container[iMGlevel], config,
                 iMGlevel, NO_RK_ITER, RUNTIME_REACTIVE_SYS, false);
  }
 }
```

where `RUNTIME_REACTIVE_SYS` is a marker of the type of governing equations. `Integration_Preprocessing` allocates the class **CMultiGridIntegration** whose constructor sets the variables that check the convergence.

`Numerics_Preprocessing` instead allocates the classes that will compute the residual for all terms(convective, viscous and source).

Inside the function `StartSolver()` we call several functions to simulate effectively the chosen problem.

```
  while ( ExtIter < config_container[ZONE_0]->GetnExtIter() ) {

  /*--- Perform some external iteration preprocessing. ---*/

  PreprocessExtIter(ExtIter);

  /*--- Perform a single iteration of the chosen PDE solver. ---*/

  if (!fsi) {

  /*--- Perform a dynamic mesh update if required. ---*/

   DynamicMeshUpdate(ExtIter);

  /*--- Run a single iteration of the problem
        (mean flow, wave, heat,...). ---*/

   Run();

  /*--- Update the solution for dual time stepping strategy ---*/

   Update();

  }
```

```
 else {
  Run();     // In the FSIDriver case, mesh and solution updates are
                    already included into the Run function
 }
 ...

 /*--- If the convergence criteria has been met,
       terminate the simulation. ---*/

 if (StopCalc) break;

 ExtIter++;

 }
```

`PreprocessExtIter(ExtIter)` calls `SetExtIter(ExtIter)` and `SetInitialCondition` to set respectively the maximum number of iteration for the external loop and the initial condition in case of an unsteady simulation.

Let us report some of the routines called by the function `Run`

```
 for (iZone = 0; iZone < nZone; iZone++)
   iteration_container[iZone]->
   Preprocess(output, integration_container, geometry_container,
              solver_container, numerics_container, config_container,
              surface_movement, grid_movement, FFDBox, iZone);
 ...
 if (unsteady)
   nIntIter = config_container[MESH_0]->GetUnst_nIntIter();
 else
   nIntIter = 1;

 for (IntIter = 0; IntIter < nIntIter; IntIter++) {
   ...
   for (iZone = 0; iZone < nZone; iZone++) {
     config_container[iZone]->SetIntIter(IntIter);

     iteration_container[iZone]->
     Iterate(output, integration_container, geometry_container,
             solver_container, numerics_container, config_container,
             surface_movement, grid_movement, FFDBox, iZone);
   }

   /*--- Check convergence in each zone --*/
   checkConvergence = 0;
   for (iZone = 0; iZone < nZone; iZone++)
    checkConvergence += (int)
        integration_container[iZone][FLOW_SOL]->GetConvergence();

   /*--- If convergence was reached in every zone --*/
   if (checkConvergence == nZone) break;
 }
}
```

The function `Iterate` calls `MultiGrid_Iteration` to execute an iteration of the full approximation scheme

```
/*--- Solve the Euler, Navier-Stokes or Reynolds-averaged Navier-Stokes
    (RANS) equations (one iteration) ---*/
if (config_container[val_iZone]->GetKind_Solver() == REACTIVE_EULER ||
    config_container[val_iZone]->GetKind_Solver() == REACTIVE_NAVIER_STOKES)
    integration_container[val_iZone][REACTIVE_SOL]->
    MultiGrid_Iteration(geometry_container, solver_container,
                        numerics_container, config_container,
                        RUNTIME_REACTIVE_SYS, IntIter, val_iZone);
else
 integration_container[val_iZone][FLOW_SOL]->
 MultiGrid_Iteration(geometry_container, solver_container,
                     numerics_container, config_container,
                     RUNTIME_FLOW_SYS, IntIter, val_iZone);
```

which calls `MultiGrid_Cycle` to compute the `Space_Integration` (suitable functions of the class **CSolver** for computing residuals and applying boundary conditions) and the `Time_Integration`(function of the class **CSolver** depending on the time marching scheme) and eventually get the prolungated solution.

```
for (iPreSmooth = 0; iPreSmooth < config[iZone]->GetMG_PreSmooth(iMesh);
    iPreSmooth++) {

 switch (config[iZone]->GetKind_TimeIntScheme()) {
 case RUNGE_KUTTA_EXPLICIT: iRKLimit = config[iZone]->GetnRKStep(); break;
 case EULER_EXPLICIT: case EULER_IMPLICIT: iRKLimit = 1; break; }

 for (iRKStep = 0; iRKStep < iRKLimit; iRKStep++) {
   ...

   /*--- Space integration ---*/

   Space_Integration(geometry[iZone][iMesh], solver_container[iZone][iMesh],
                     numerics_container[iZone][iMesh][SolContainer_Position],
                     config[iZone], iMesh, iRKStep, RunTime_EqSystem);

   /*--- Time integration, update solution using the old solution plus the
       solution increment ---*/

   Time_Integration(geometry[iZone][iMesh], solver_container[iZone][iMesh],
                    config[iZone], iRKStep, RunTime_EqSystem, Iteration);

   ...

   }
}
...
if ( (iMesh < config[iZone]->GetnMGLevels() && ((Iteration >=
    config[iZone]->GetnStartUpIter()) || startup_multigrid)) ) {

  /*--- Compute $r_k = P_k + F_k(u_k)$ ---*/

  solver_container[iZone][iMesh][SolContainer_Position]->
```

```
Preprocessing(geometry[iZone][iMesh], solver_container[iZone][iMesh],
              config[iZone], iMesh, NO_RK_ITER, RunTime_EqSystem, false);

Space_Integration(geometry[iZone][iMesh], solver_container[iZone][iMesh],
                  numerics_container[iZone][iMesh][SolContainer_Position],
                  config[iZone], iMesh, NO_RK_ITER, RunTime_EqSystem);

SetResidual_Term(geometry[iZone][iMesh],
                 solver_container[iZone][iMesh][SolContainer_Position]);

/*--- Compute $r_(k+1) = F_(k+1)(I^(k+1)_k u_k)$ ---*/

SetRestricted_Solution( RunTime_EqSystem,
                        solver_container[iZone][iMesh][SolContainer_Position],
                        solver_container[iZone][iMesh+1][SolContainer_Position],
                        geometry[iZone][iMesh], geometry[iZone][iMesh+1],
                        config[iZone]);

solver_container[iZone][iMesh+1][SolContainer_Position]->
Preprocessing(geometry[iZone][iMesh+1], solver_container[iZone][iMesh+1],
              config[iZone], iMesh+1, NO_RK_ITER, RunTime_EqSystem,
              false);

Space_Integration(geometry[iZone][iMesh+1],
                  solver_container[iZone][iMesh+1],
                  numerics_container[iZone][iMesh+1][SolContainer_Position],
                  config[iZone], iMesh+1, NO_RK_ITER, RunTime_EqSystem);

/*--- Compute $P_(k+1) = I^(k+1)_k(r_k) - r_(k+1) ---*/

SetForcing_Term(solver_container[iZone][iMesh][SolContainer_Position],
                solver_container[iZone][iMesh+1][SolContainer_Position],
                geometry[iZone][iMesh], geometry[iZone][iMesh+1],
                config[iZone], iMesh+1);

/*--- Recursive call to MultiGrid_Cycle ---*/

for (unsigned short imu = 0; imu <= RecursiveParam; imu++) {
 if (iMesh == config[iZone]->GetnMGLevels()-2)
  MultiGrid_Cycle(geometry, solver_container, numerics_container,
                  config, iMesh+1, 0, RunTime_EqSystem,
                  Iteration, iZone);
 else
  MultiGrid_Cycle(geometry, solver_container, numerics_container,
                  config, iMesh+1, RecursiveParam, RunTime_EqSystem,
                  Iteration, iZone);
}

/*--- Compute prolongated solution, and smooth the correction $u^(new)_k
    = u_k + Smooth(I^k_(k+1)(u_(k+1)-I^(k+1)_k u_k))$ ---*/

GetProlongated_Correction(RunTime_EqSystem,
                          solver_container[iZone][iMesh][SolContainer_Position],
                          solver_container[iZone][iMesh+1][SolContainer_Position],
                          geometry[iZone][iMesh],
                          geometry[iZone][iMesh+1],
                          config[iZone]);
```

```
  SmoothProlongated_Correction(RunTime_EqSystem,
                               solver_container[iZone][iMesh][SolContainer_Position],
                               geometry[iZone][iMesh],
                               config[iZone]->GetMG_CorrecSmooth(iMesh),
                               1.25, config[iZone]);

  SetProlongated_Correction(solver_container[iZone][iMesh][SolContainer_Position],
                            geometry[iZone][iMesh], config[iZone], iMesh);
```

The function `Update` calls the function `Update` of the class **CMeanFlowIteration**

```
for(iZone = 0; iZone < nZone; iZone++)
 iteration_container[iZone]->Update(output, integration_container,
                                    geometry_container, solver_container,
                                    numerics_container, config_container,
                                    surface_movement, grid_movement,
                                    FFDBox, iZone);
```

which sets the solution at time $n-1$ and n for the dual time solver.

Let us report the main subroutines of the function `Postprocessing` inside the class **CFluidDriver**

```
  for (iZone = 0; iZone < nZone; iZone++) {
    Numerics_Postprocessing(numerics_container[iZone],
                            solver_container[iZone],
                            geometry_container[iZone],
                            config_container[iZone]);
    delete [] numerics_container[iZone];
  }
  delete [] numerics_container;
  if (rank == MASTER_NODE) cout << "Deleted CNumerics container." << endl;

  for (iZone = 0; iZone < nZone; iZone++) {
    Integration_Postprocessing(integration_container[iZone],
                               geometry_container[iZone],
                               config_container[iZone]);
    delete [] integration_container[iZone];
  }
  delete [] integration_container;
  if (rank == MASTER_NODE) cout << "Deleted CIntegration container." <<
      endl;

  for (iZone = 0; iZone < nZone; iZone++) {
    Solver_Postprocessing(solver_container[iZone],
                          geometry_container[iZone],
                          config_container[iZone]);
    delete [] solver_container[iZone];
  }
  delete [] solver_container;
  if (rank == MASTER_NODE) cout << "Deleted CSolver container." << endl;
```

```
for (iZone = 0; iZone < nZone; iZone++) {
  delete iteration_container[iZone];
}
delete [] iteration_container;
if (rank == MASTER_NODE) cout << "Deleted CIteration container." << endl;
```

All the functions for `Postprocessing` are simply delegated to delete the instances of the classes created in the corresponding `Preprocessing`.

Chapter 5

Numerical Results

In this section we report the main results from the numerical simulations performed during this work. Its starts from three basic test-cases for non reacting flows in order to validate the code for multispecies simulations (inviscid bump, diffusion inside a channel and laminar flat plate) and develops the discussion into more complex conditions (combustion inside aerospace engine and re-entry case)

5.1 Inviscid Bump

The first case to test the correct behaviour of the code is represented by the inviscid bump in a channel: this example uses a 2D geometry that features a circular bump along the lower wall. This kind of flow is one of the benchmark problems for subsonic inlet and outlet boundary conditions in SU2 and therefore we can employ the already tested mesh.
In detail the channel is of length 3 m with a height of 1 m and a circular bump centered along the lower wall with height 0.1 m, while the mesh is composed of quadrilaterals with 256 nodes along the length of the channel and 128 nodes along the height.

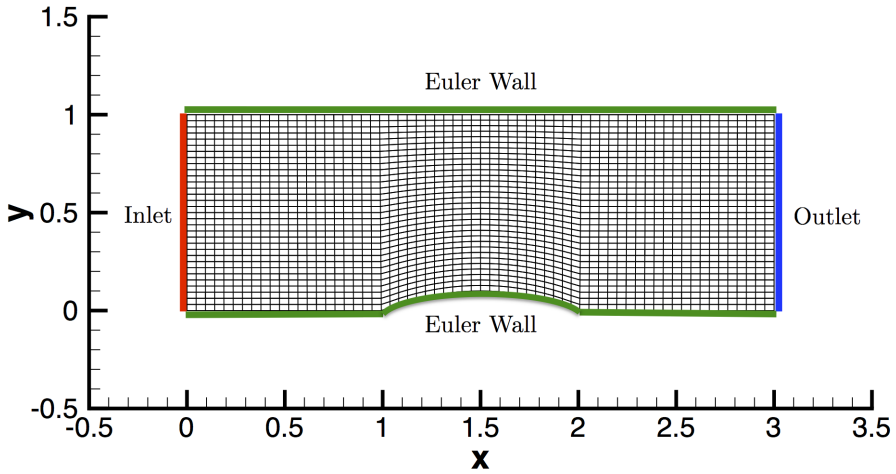


Figure 5.1: Computational mesh with highlighted boundary conditions

At inlet boundary **total boundary conditions** are imposed with a total temperature of 288.6 K and a total pressure of 102010.0 Pa, while the static pressure imposed at outlet is equal to 101325.0 Pa.

Eventually the species chosen for the simulation is O_2. Let us compare the results at steady state with a second order scheme:

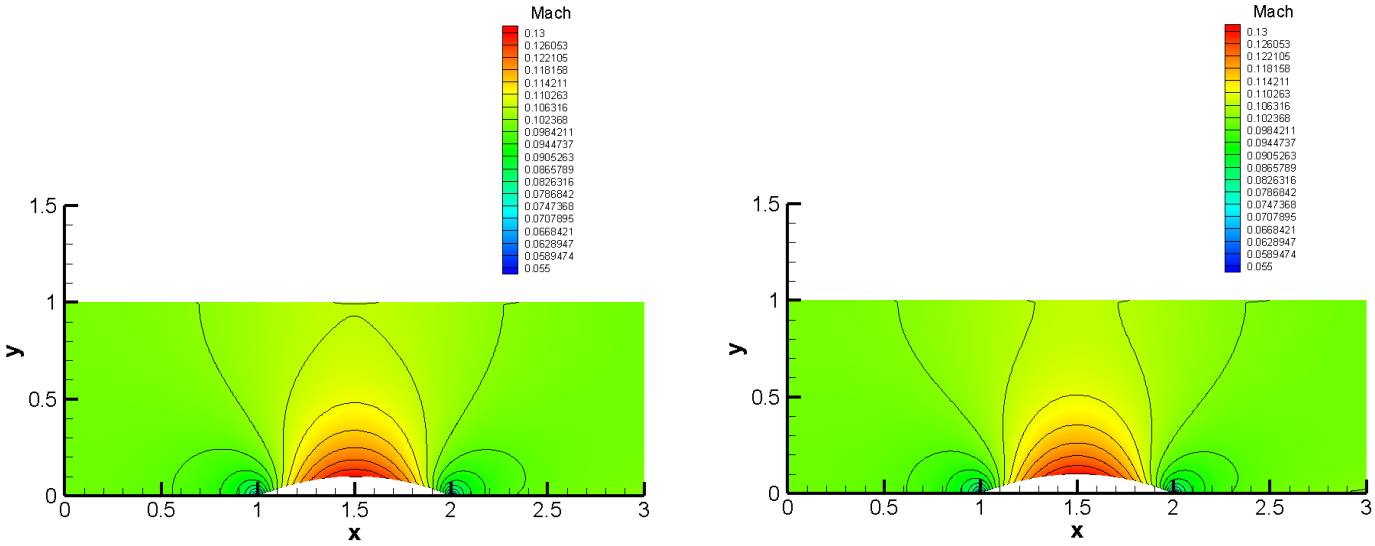


Figure 5.2: Comparison of Mach number contours between my version (left) and the original one (right)

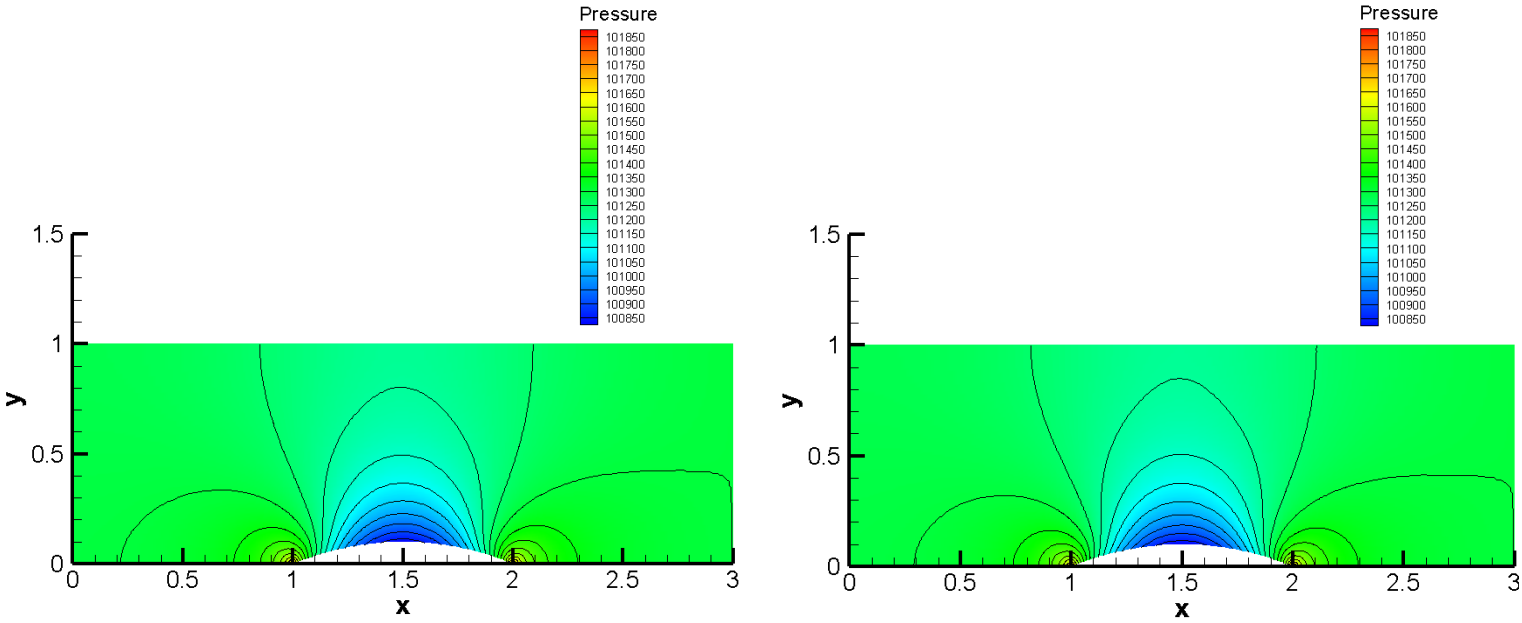


Figure 5.3: Comparison of pressure contours between my version (left) and the original one (right)

The results obtained with my version are in good agreement with respect to the original one and the expected symmetry due to the geometry and the model is well caught.

Finally another simulation involving three species (78% N_2, 20% O_2 and 2% CO_2) has been performed in order to verify the behaviour in case of multispecies flows.
Let us report the results at steady state:

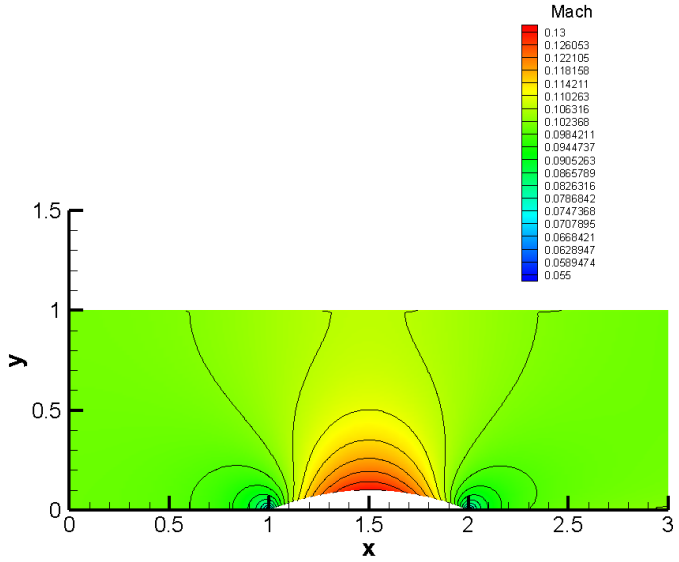


Figure 5.4: Mach number contour for multispecies simulation

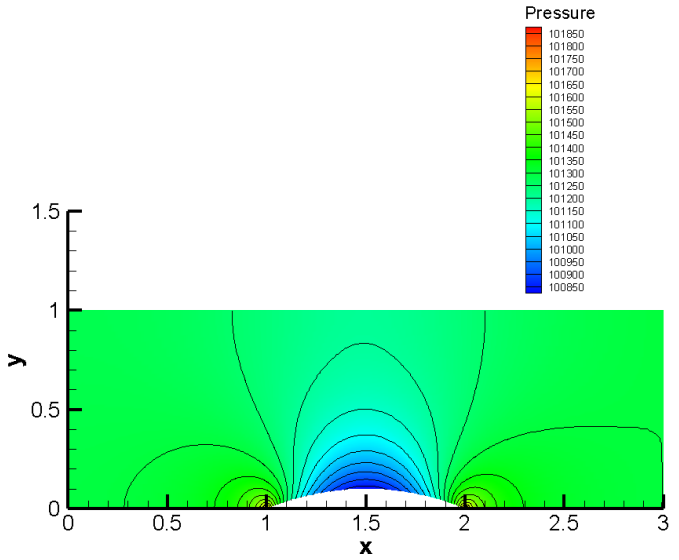


Figure 5.5: Pressure contour for multispecies simulation

Also in this case the simulation shows the expected symmetric behaviour with values very similar to the previous simulations since the gas constant values are not so different.

5.2 Diffusion in a channel

The second test case regards the diffusion of two species inside a channel in order to verify if the modifications of Stefan-Maxwell equations previously described work properly. The channel is long 1 m and height 0.06 m and the mesh consists of 120 x 40 nodes.
As initial condition the channel is full of CO so that we can verify that the model is able to eliminate a species that is no more injected and does not react.
Moreover the species denoted as CO_2 has the same molar mass of O_2 in order to avoid pressure gradients at inlet so that streamlines do not curve along the domain.
Eventually at inlet we inject O_2 in the upper part and CO_2 in the lower part with a total temperature of 500 K and a total pressure of 102 000 Pa, while at outlet we set a static pressure of101 325 Pa.

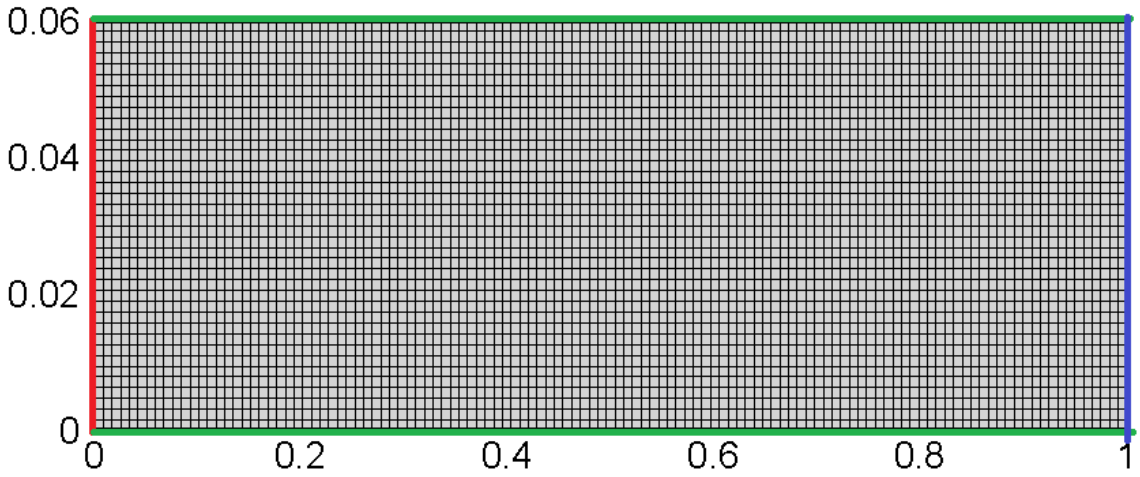


Figure 5.6: Mesh (stretched in y for visibility) and boundary conditions: Inlet, Outlet, No-slip walls

These are the contours of species mass fractions at steady state:

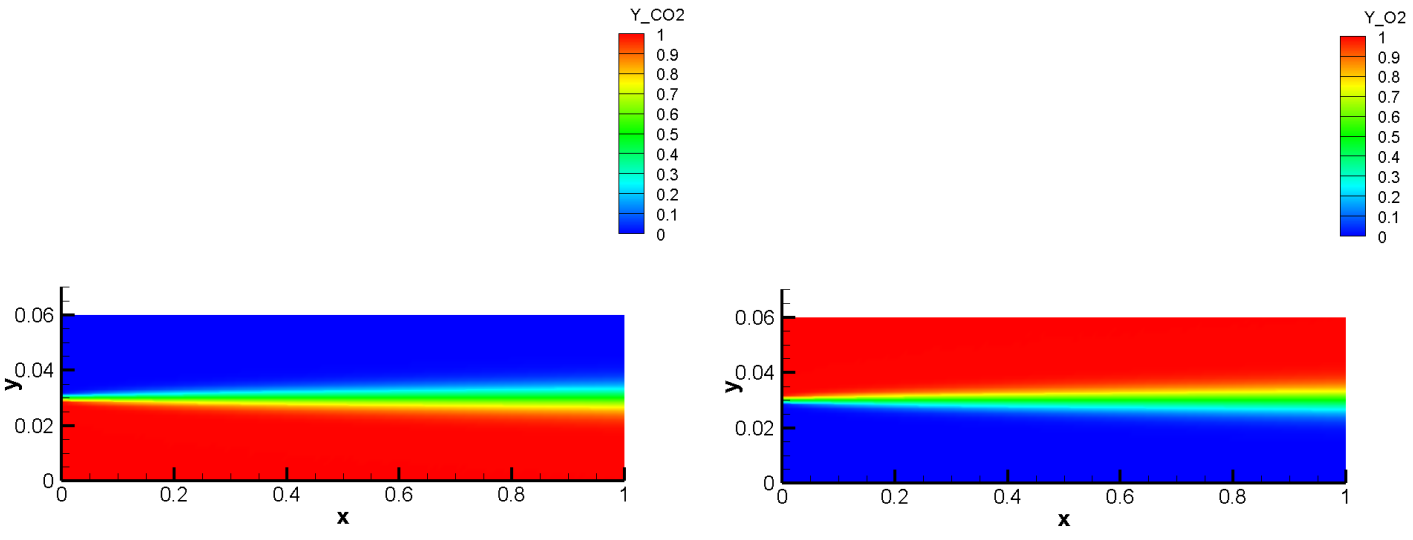


Figure 5.7: Steady state distribution of CO_2 mass fraction (left) and O_2 mass fraction (right)

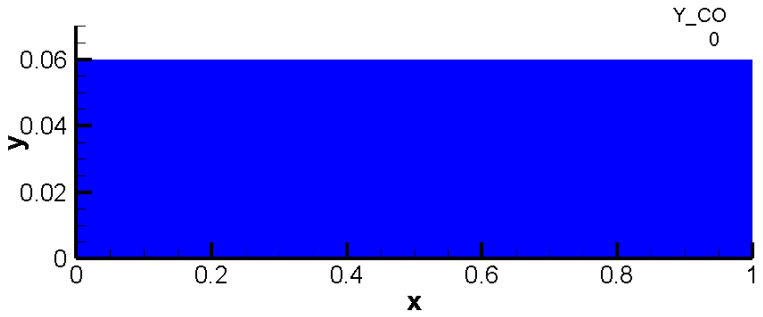


Figure 5.8: Steady state distribution of CO mass fraction

Moreover we report the profile of CO_2 and O_2 mass fractions at $x = 0.8\,\mathrm{m}$

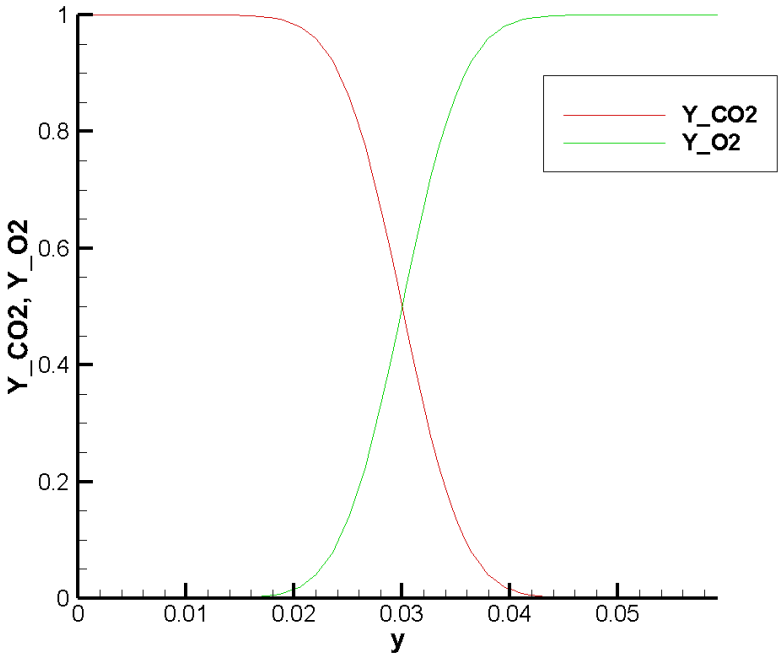


Figure 5.9: Steady state profile of CO_2 and O_2 mass fractions at $x = 0.8\,\mathrm{m}$

The diffusion phenomenon is very well caught at interface as it can be seen by the symmetry of the profile: the two species diffuse from $y \approx 0.02\,\mathrm{m}$ up to $y \approx 0.04\,\mathrm{m}$ and at $y = 0.03\,\mathrm{m}$ we find half O_2 and half CO_2.
Moreover there are no residuals of CO as expected.

5.3 Laminar Flat Plate

Another interesting test case regards the capability to catch boundary layer; therefore a comparison analogous to the one previously described for the bump is performed.
The computational mesh for the flat plate is composed of quadrilaterals with 65 nodes in both the x and y directions. The flat plate is along the lower boundary of the domain (y = 0) starting at x = 0 m and is of length 0.3048 m.

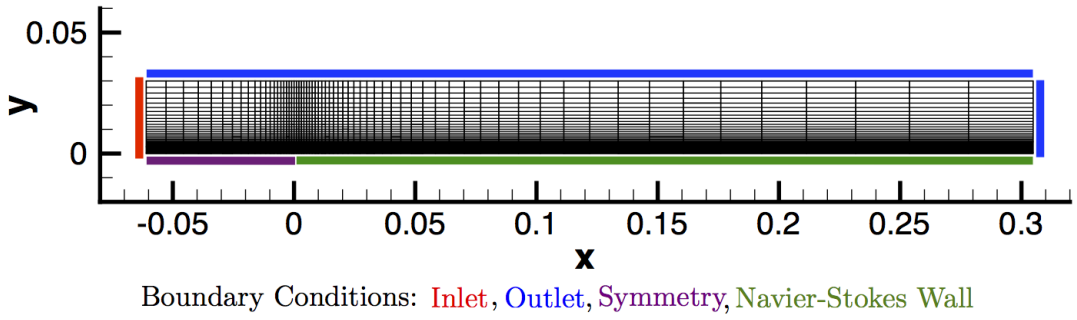


Figure 5.10: Computational mesh with highlighted boundary conditions

At inlet boundary total conditions are imposed with a total temperature of 300 K and a total pressure of 100 000 Pa, while the static pressure imposed at outlet is equal to 97 250 Pa.
As in the previous case we choose O_2 to compare the results at steady state between the original code and my version.

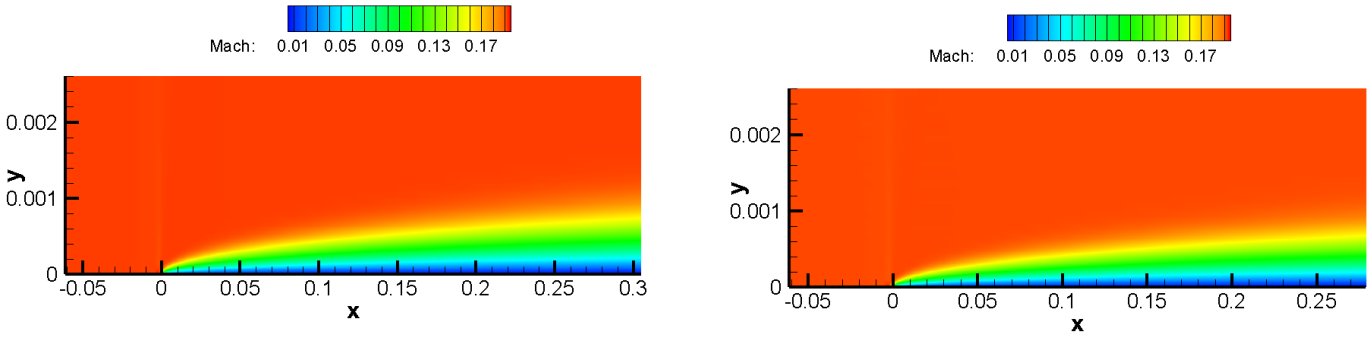


Figure 5.11: Comparison of Mach number contours between my version (left) and the original one (right)

As it can be noticed the two simulations match well: the development of the boundary layer is present in the results with analogous height.

Eventually we perform a simulation with three species taking the same composition described for the bump, i.e 78% N_2, 20% O_2 and 2% CO_2, and we show the results at steady state:

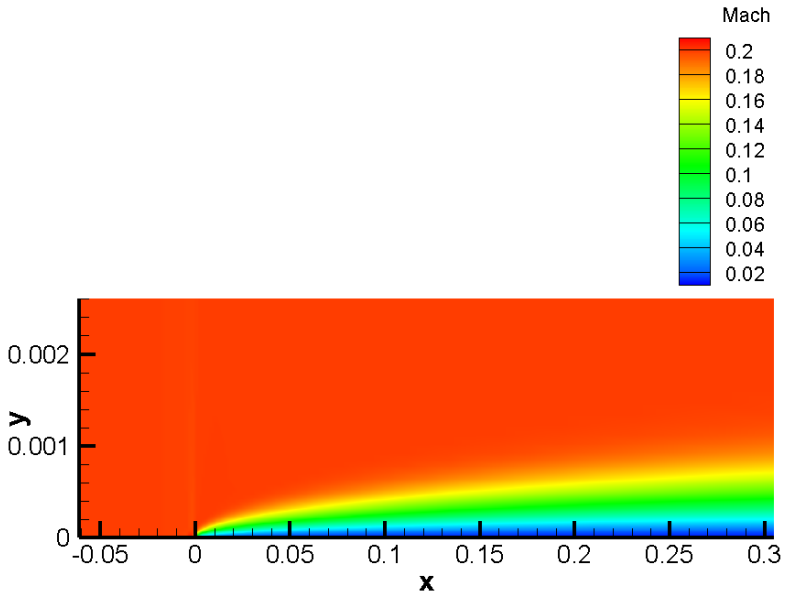


Figure 5.12: Mach number contour for flat plate simulation

The boundary layer is present even in this case and in order to support the results we compare them with the well known Blasius profile ([28]):

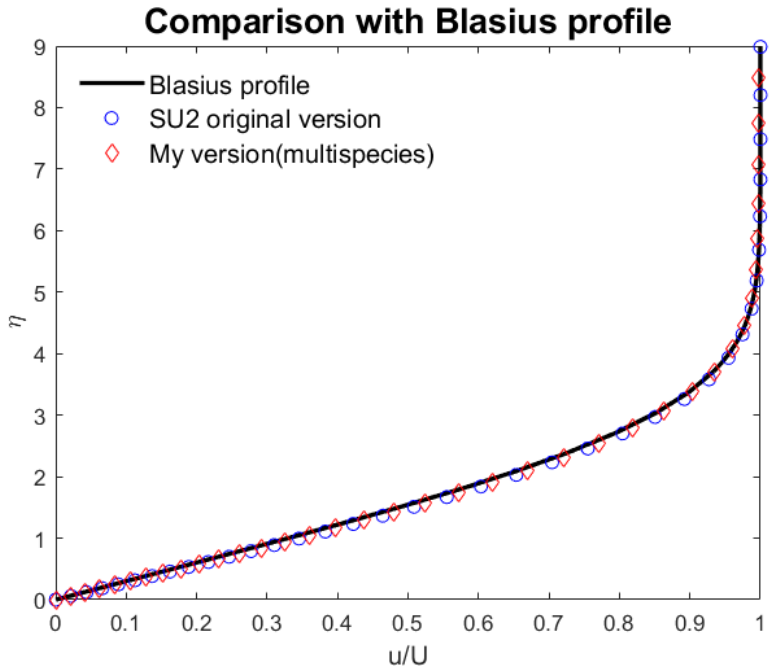


Figure 5.13: Comparison with Blasius profile at $x = 0.3048\,\mathrm{m}$(O_2 for original version of SU2)

As it can be noticed in Figure (5.13) the behaviour matches nearly perfectly not only the original version of SU2 but also the analytical solution of Blasius equation.

5.4 Combustion and Hybrid Rocket Engine

A good starting point to define **combustion** is provided by [16] as "*a rapid oxidation generating heat or both light and heat*".
Combustion can occure in either a **flame** or **nonflame** mode and flames are categorized as being either **premixed flames** or **nonpremixed(diffusion) flames**. The two classes of flames are related to the state of mixedness of reactants as suggested by the names: in a premixed flame the fuel and the oxidizer are mixed at a molecular level before the occurrence of any chemical reaction, while in a diffusion flame the reactants are initially separated and reaction occurs only at the interface between the fuel and the oxidizer where also the mixing takes place.

Historically within the framework of space technology two kind of engines have been used: solid rocket engine and liquid rocket engine.
Liquid-propellant rockets are in general more efficient because the high density allows the volume of propellant tanks to be low and therefore the use of low-mass propellant tanks results in a high **mass ratio** (a measure of the efficiency of rockets which describes how much more massive the vehicle is with propellant than without) and allows to obtain higher velocity changes; on the other hand solid-propellant rockets can remain in storage for a long time without too much propellant degradation and so provide high thrust for relatively low cost.

Hybrid rocket engine try to conjugate precision and safety of liquid-fuelled rockets with design simplicity and low cost of solid-fuelled rockets.
The main characteristic of a hybrid rocket engine is that oxidizer and fuel are in different phases and therefore the resulting motor shows features of both liquid and solid propellants systems; in the typical configuration the fuel is a solid of cylindrical shape while the oxidizer is injected into its port as a liquid spray or a gas.

Chemistry

In order to choose the correct chemical model, an assumption has to be made about pyrolisis products of the considered fuel (HTPB): as reported in [1], according to [8] and [9] the main product during pyrolisis process is gaseous 1,3 butadiene C_4H_6. The first approach is to implement a very simple chemical model for the combustion of butadiene gas and as shown in [6], it is possible to introduce a two step, five species combustion model (Venkateswaran - Merkle model):

$$1.\ C_4H_6 + 3.5O_2 \longrightarrow 4CO + 3H_2O$$

$$2.\ CO + 0.5O_2 \longleftrightarrow CO_2$$

The mass production terms are computed as follows:

$$\begin{aligned}
&\dot{\omega_{C_4H_6}} = -M_{C_4H_6} k_{f,1} [C_4H_6][O_2] \\
&\dot{\omega_{O_2}} = M_{O_2} \left\{ -3.5 k_{f,1} [C_4H_6][O_2] - 0.5 \left(k_{f,2} [CO][O_2]^{0.5} - k_{b,2} [CO_2] \right) \right\} \\
&\dot{\omega_{CO}} = M_{CO} \left\{ 4 k_{f,1} [C_4H_6][O_2] - \left(k_{f,2} [CO][O_2]^{0.5} - k_{b,2} [CO_2] \right) \right\} \\
&\dot{\omega_{H_2O}} = 3 M_{H2_O} k_{f,1} [C_4H_6][O_2] \\
&\dot{\omega_{CO_2}} = M_{CO_2} \left(k_{f,2} [CO][O_2]^{0.5} - k_{b,2} [CO_2] \right)
\end{aligned}$$

where

$$k_{f,1}(T) = 8.80 \cdot 10^{11} \exp(-30000/(R_u T))$$
$$k_{f,2}(T) = 10^{14.6} \exp(-40000/(R_u T))$$
$$k_{b,2}(T) = 5.00 \cdot 10^{8} \exp(-40000/(R_u T))$$

with $R_u = 1.985\,75\,\mathrm{cal/(mol \cdot K)}$.

Let us now briefly describe the addressed geometry: the chamber is described as a 2D domain with an axial inlet for the oxidizer and a transverse inlet for fuel which extends from 0.025 m to 0.075 m along x direction. The chamber is 0.125 m long and 0.006 0.006 m height.
For what concerns the computational mesh we rely on [1] and therefore we know that a mesh with 100 x 80 elements is sufficient to reach convergence. Anyway in order to catch even better the behaviour near the walls we employ a tapered mesh with 90 cells in y direction and more specifically 20 nodes from 0 m to 0.001 m and from 0.005 m to 0.006 m

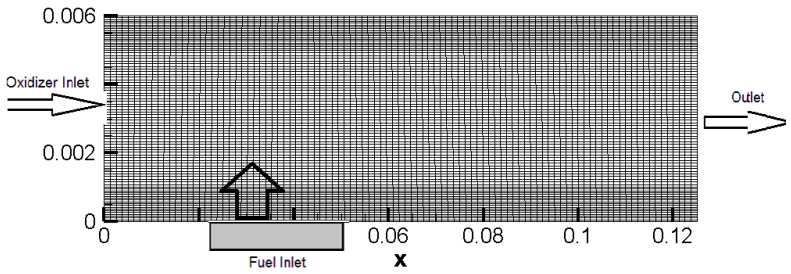


Figure 5.14: Computational mesh for chamber

Let us summarize now the employed boundary conditions: we underline that the walls are treated as isothermal no-slip walls.

Table 5.1: Boundary conditions

Boundary	u [m/s]	v [m/s]	T [K]	p [Pa]
Oxidizer Inlet	20	0	300	-
Fuel Inlet	0	0.86	800	-
Upper Wall	0	0	300	-
Lower Wall - Pre-Inlet	0	0	300	-
Lower Wall - Post-Inlet	0	0	600	-
Outlet	-	-	-	101325

Figure 5.15: Summary of boundary conditions for combustion

Let us report now some important considerations about convergence and CFL condition: simulations that involve chemistry, in particular the one with combustion, are very stiff because reactions lead to a large heat release and subsequent density changes and large accelerations in the flow.
For this reason the value of the Courant-Friedrichs-Levy (CFL) number must be kept low and a good way to reach a stable solution is employing a two-step procedure: we begin

by solving the flow with reactions disabled and when the pattern has been established we re-enable the reactions and continue the calculation.

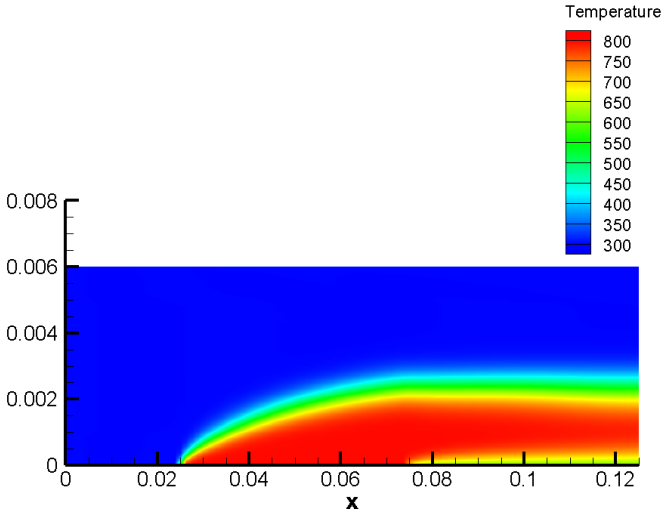


Figure 5.16: Contour plot of the temperature without chemistry

In all our simulations we set CFL equal to 0.2 and as default parameter convergence is reached for a residual norm below 10^{-4}.

These are the results for what concerns the Venkateswaran - Merkle model:

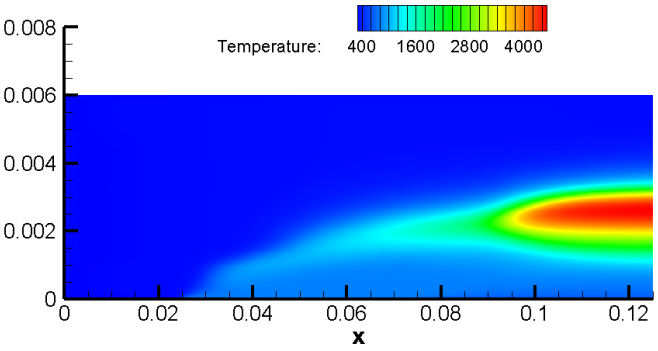


Figure 5.17: Temperature contour at steady state for combustion chamber (Merkle)

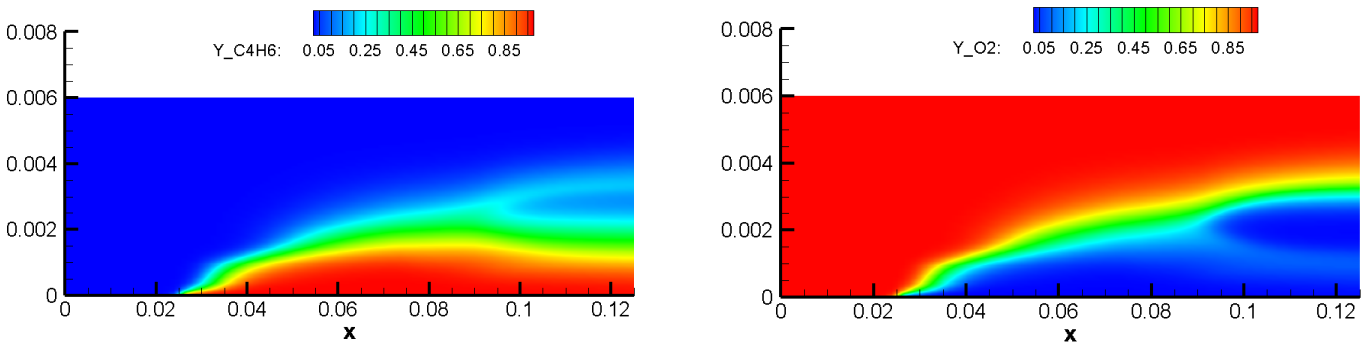


Figure 5.18: C_4H_6 (left) and O_2 (right) contours at steady state (Merkle)

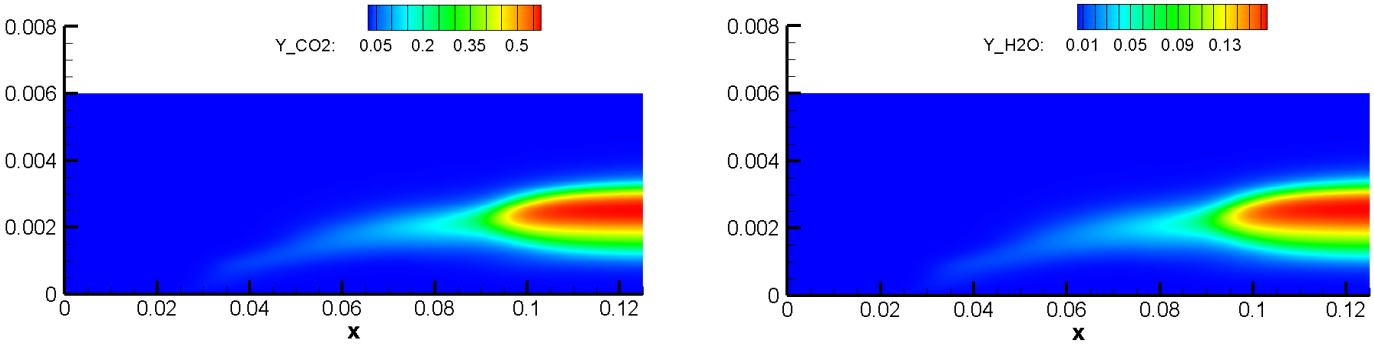


Figure 5.19: CO_2 (left) and H_2O (right) contours at steady state (Merkle)

Reactant distribution is coherent with inlet positioning: fuel (C_4H_6) remains in lower region and then it is burnt, while oxidizer (O_2) remains in the top region and reacts in the flame region.
Products distribution is coherent with peak temperature region, were chemical reactions are more intense.
Finally the peak temperature is significantly high but, as explained by [1], this is due to the simplicity of the model which does not allow a correct and quantitative study of combustion processes in hybrid rocket engine.

A more detailed four reactions, six species scheme from Jones and Linstedt (JL) [7] has been then considered:

$$1.\ C_4H_6 + 2O_2 \longrightarrow 4CO + 3H_2$$

$$2.\ C_4H_6 + 4H_2O \longrightarrow 4CO + 7H_2$$

$$3.\ CO + H_2O \longleftrightarrow CO_2 + 3H_2$$

$$4.\ H_2 + 0.5O_2 \longleftrightarrow H_2O$$

with the following parameters

N. reaction	A_r	E_r	β_r	Reaction rate	a	b
1	9.44e+10	30000	0	$k_{f,1}[C_4H_6]^a[O_2]^b$	0.5	1.25
2	7.84e+10	30000	0	$k_{f,2}[C_4H_6]^a[O_2]^b$	1	1
3	2.75e+12	20000	0	$k_{f,3}[C_4H_6]^a[O_2]^b$	1	1
4	1.43e+17	40000	-1	$k_{f,4}[C_4H_6]^a[O_2]^b$	0.25	1.5

Table 5.1: Parameters of JL (4 reactions) chemical scheme: units are $\mathrm{cm}, \mathrm{mol}, \mathrm{s}, \mathrm{cal}$

Herein we report the main results:

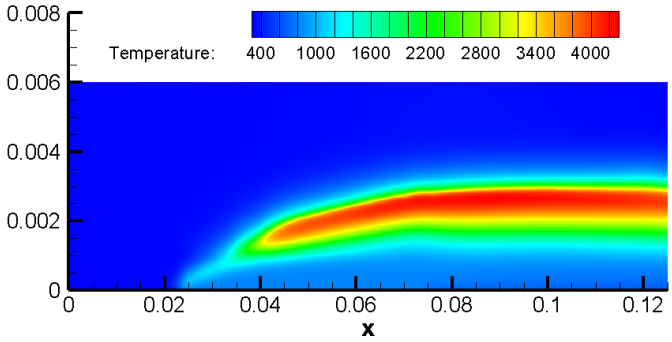


Figure 5.20: Temperature contour for combustion chamber (JL 4 reactions)

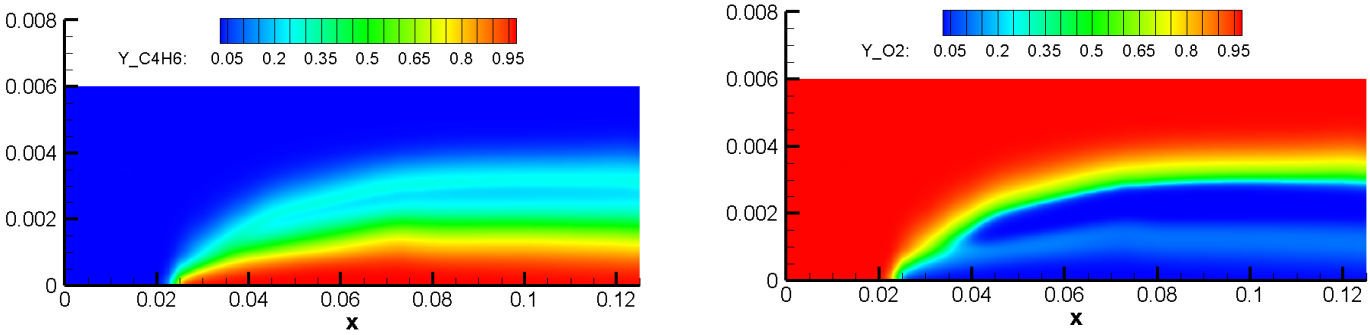


Figure 5.21: C_4H_6 (left) and O_2 (right) contours (JL 4 reactions)

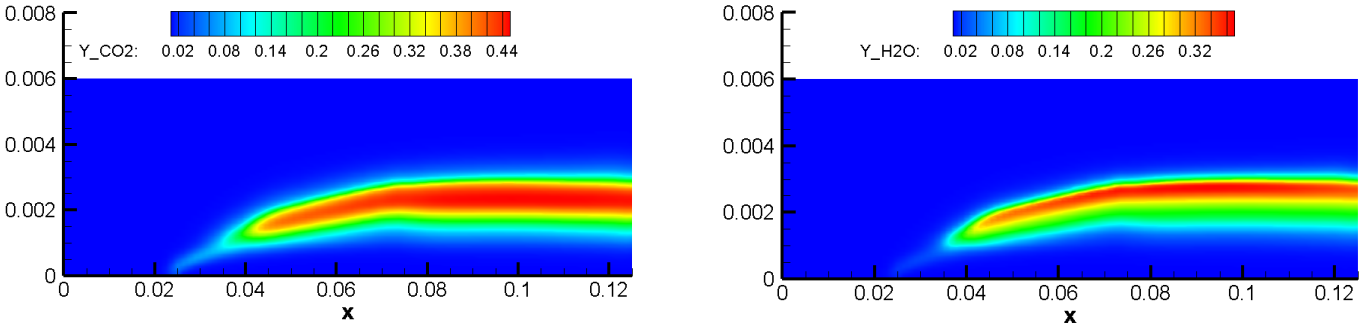


Figure 5.22: CO_2 (left) and H_2O (right) contours (JL 4 reactions)

The qualitative behaviour previously described is caught even with this model; anyway the peak temperature is still beyond 4000 K: as explained in [1], this scheme with 4 reactions is not a suitable candidate for quantitative analysis because it lacks of any strong energy-absorbing chemical process as shown by the high presence of H_2O.

Therefore a better compromise between number of chemical species involved and computational cost is the six reactions, nine species model proposed again by Jones and Linstedt [7]:

1. $C_4H_6 + 2O_2 \longrightarrow 4CO + 3H_2$
2. $C_4H_6 + 4H_2O \longrightarrow 4CO + 7H_2$

3. $CO + H_2O \longleftrightarrow CO_2 + 3H_2$

4. $H_2 + 0.5O_2 \longleftrightarrow H_2O$

5. $O_2+ \longleftrightarrow O + O$

6. $H_2O \longleftrightarrow OH + H$

with the following parameters

N. reaction	A_r	E_r	β_r	Reaction rate	a	b
1	9.44e+10	30000	0	$k_{f,1}[C_4H_6]^a[O_2]^b$	0.5	1.25
2	7.84e+10	30000	0	$k_{f,2}[C_4H_6]^a[O_2]^b$	1	1
3	2.75e+12	20000	0	$k_{f,3}[C_4H_6]^a[O_2]^b$	1	1
4	1.43e+17	40000	-1	$k_{f,4}[C_4H_6]^a[O_2]^b$	0.25	1.5
5	1.50e+09	113000	0	$k_{f,5}[O_2]^a$	1	
6	2.30e+22	40000	-3	$k_{f,6}[H_2O]^a$	1	

Table 5.2: Parameters of JL (6 reactions) chemical scheme: units are $\mathrm{cm}, \mathrm{mol}, \mathrm{s}, \mathrm{cal}$

Herein we report the main results:

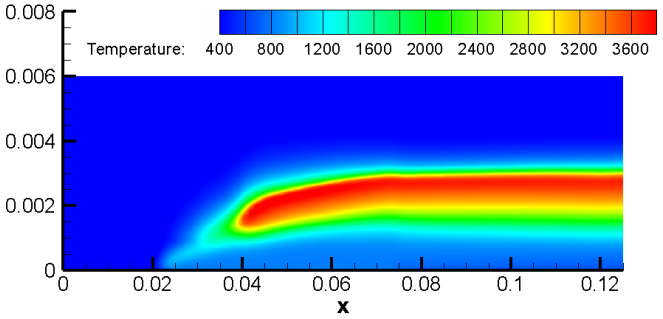


Figure 5.23: Temperature contour for combustion chamber (JL 6 reactions)

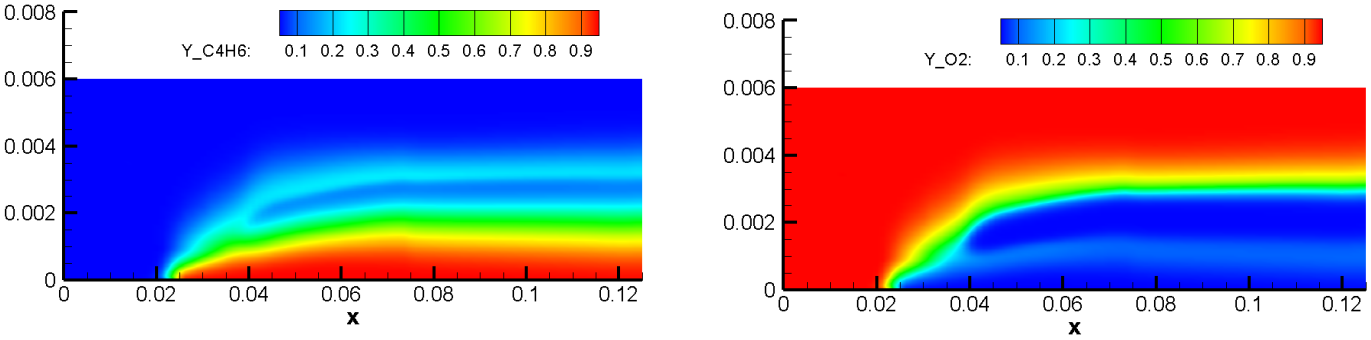


Figure 5.24: C_4H_6 (left) and O_2 (right) contours (JL 6 reactions)

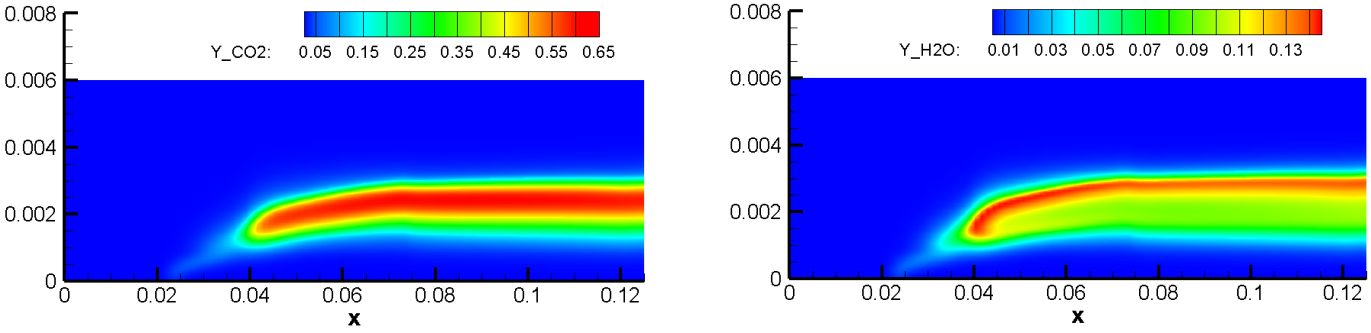


Figure 5.25: CO_2 (left) and H_2O (right) contours (JL 6 reactions)

For further considerations we report also the profile of some mass fractions at $x = 0.1\,\text{m}$ and $x = 0.120\,\text{m}$:

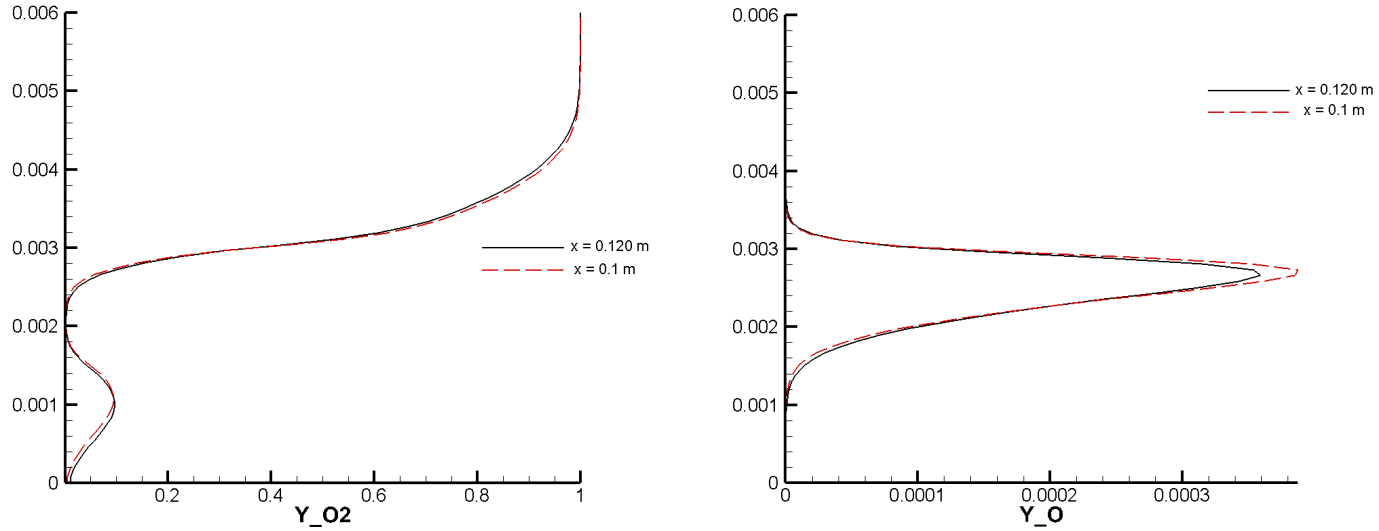


Figure 5.26: O_2 (left) and O (right) profiles (JL 6 reactions)

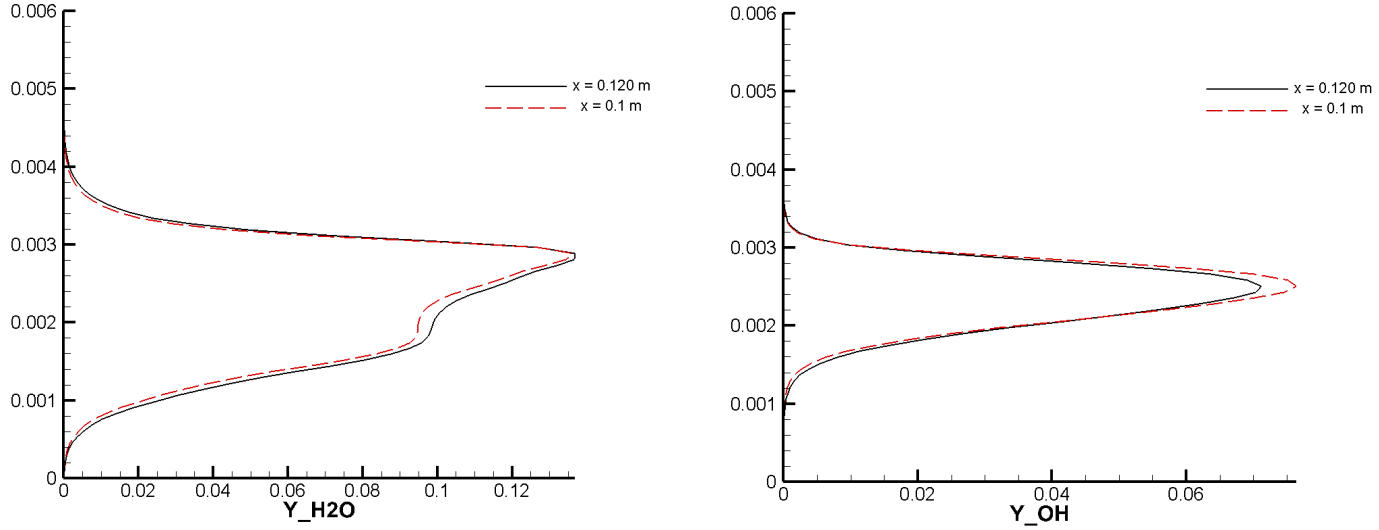


Figure 5.27: H_2O (left) and OH (right) profiles (JL 6 reactions)

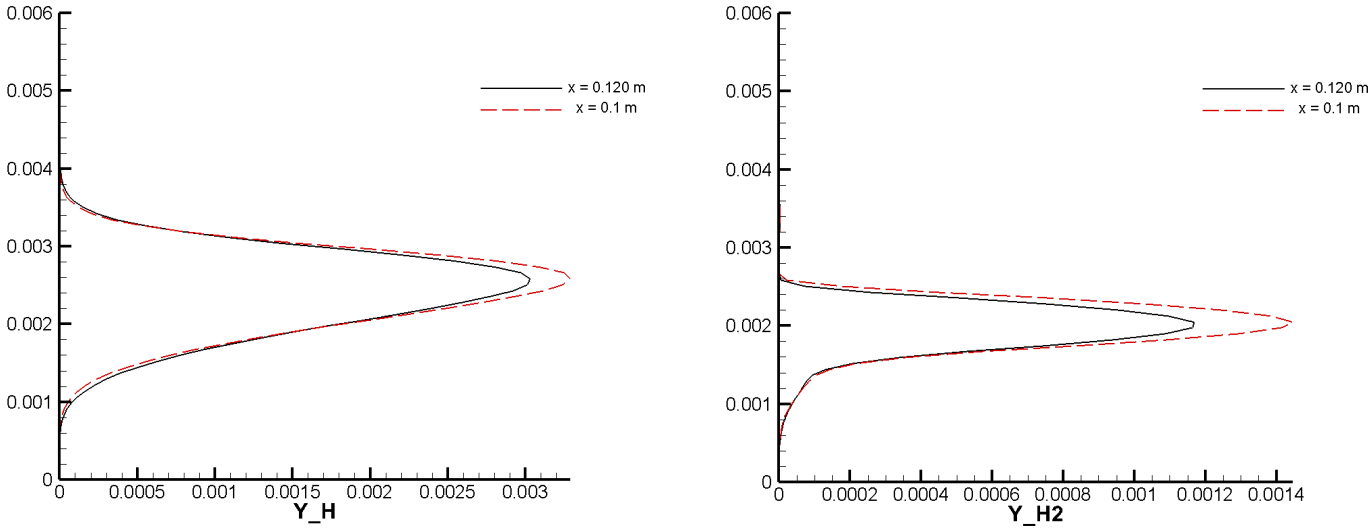


Figure 5.28: H (left) and H_2 (right) profiles (JL 6 reactions)

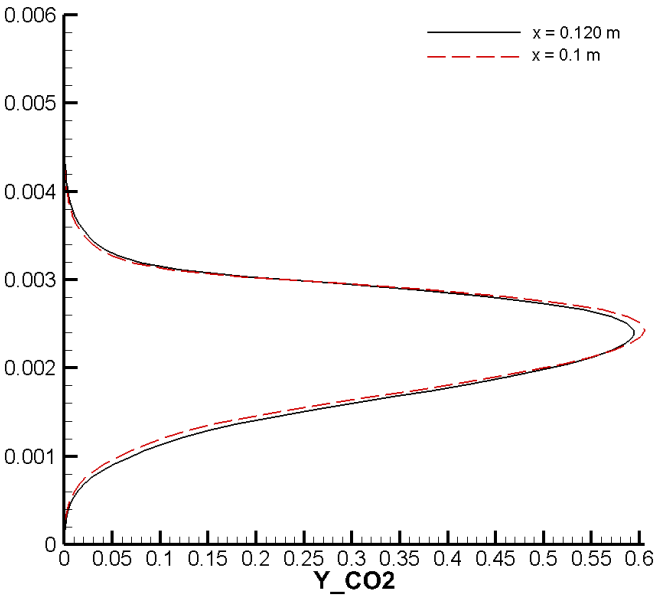


Figure 5.29: CO_2 profile (JL 6 reactions)

The peak temperature, about 3800 K, and the flame geometry are in line with literature results (see [1]).
In the highest temperature region H_2O dissociates into H and OH and O_2 dissociates in O. CO_2, which appears in the chemical model only as reaction product, is generated homogeneously along the flame. As also expected by problem physics, being H_2O a reactant in the production of CO_2, it is possible to notice a separation between the regions where mass fractions of these two species are present. The same phenomena involves also H_2 mass fraction with respect to H_2O mass fraction, being the first a reaction antagonist of the latter in all the chemical reactions involving both of them.

Moreover we report a comparison of the H_2O profile between the four reactions and the six reactions model at the end of combustion chamber where reactions are more

intense.

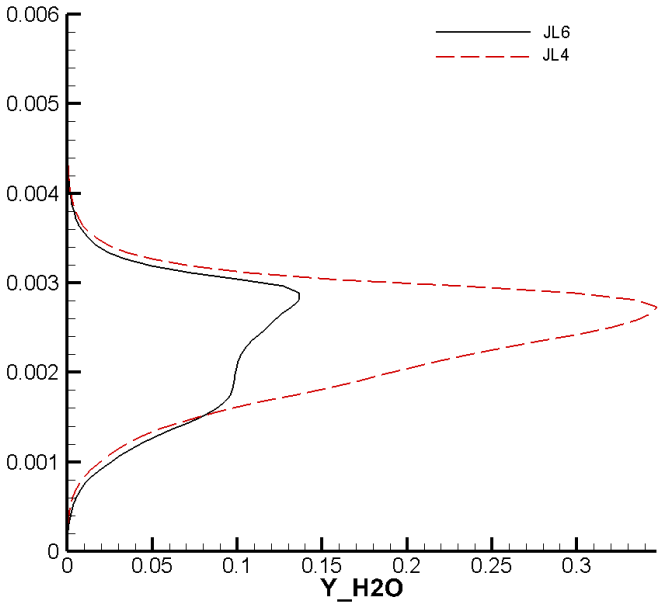


Figure 5.30: H_2O profile comparison between JL 4 reactions and JL 6 reactions

As it can be noticed the mass fraction of H_2O in case of the model with four reactions is much higher than the one obtained with the model with 6 reactions: this is probably due to the lack of the last two reactions that characterize JL 6 reactions scheme which express the water dissociation.

Eventually we perform a qualitative analysis comparing the obtained mass fractions profiles of H_2O and CO_2 with the one reported in [1]; in his simulation indeed Mazzetti adopted a turbulent regime or, more precisely, a turbulent regime where the contribution related to chemical reactions have been computed in a laminar framework.

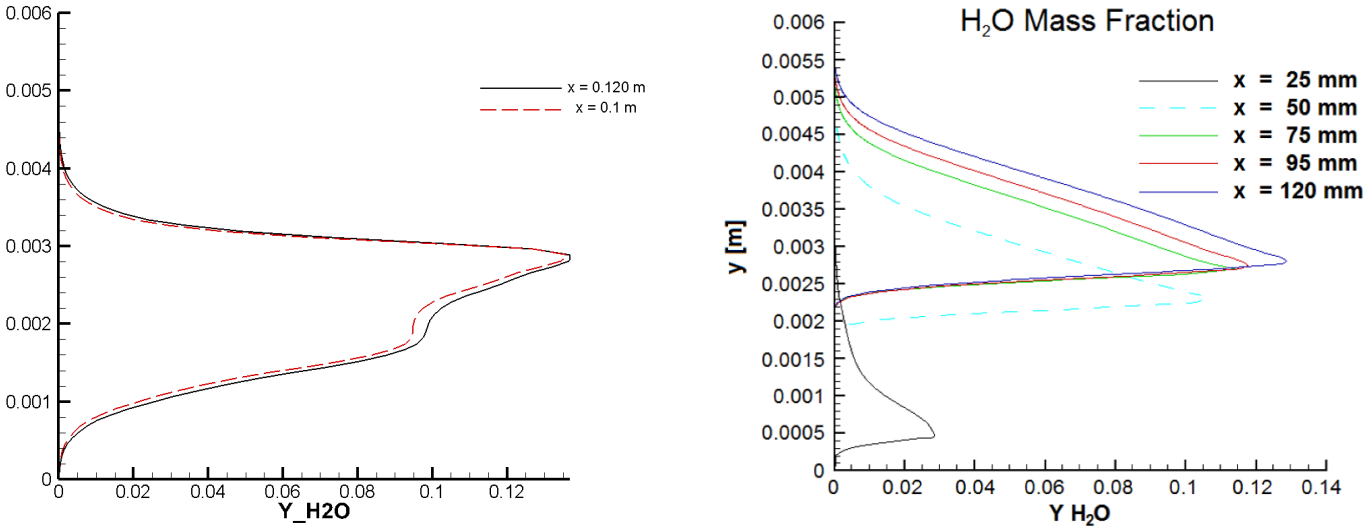


Figure 5.31: H_2O mass fraction comparison between my simulation(left) and Mazzetti(right)

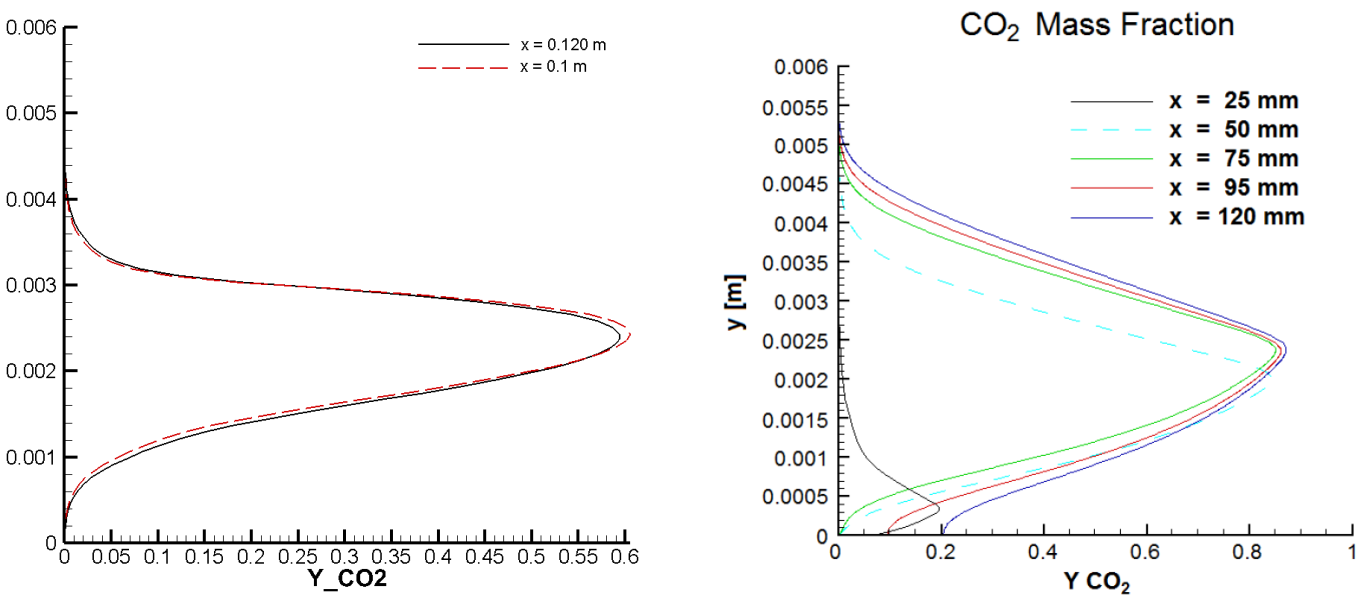


Figure 5.32: CO_2 mass fraction comparison between my simulation (left) and Mazzetti (right)

The qualitative behaviour is quite similar and the position on flame, which is detected by the portion where the species are more present, is in accordance with between the two simulations.
There are some differences from a quantitative point of view but this is probably due to the fact that Mazzetti worked in a turbulent regime.

5.5 Hypersonic flow over blunt body

The last simulation deals with a hypersonic flow over a blunt body.
The high velocities reached allow us to employ an Euler model; this case represents a good test for re-entry: indeed, since this kind of flow is characterized by hypersonic regime ($Ma >> 1$), we are in presence of shock waves.
The mesh consists of 81 nodes at inlet and 61 nodes at outlet: it is a structured grid with elements perpendicular to the arc of circumference in correspondence of Euler wall (see Figure 5.33).

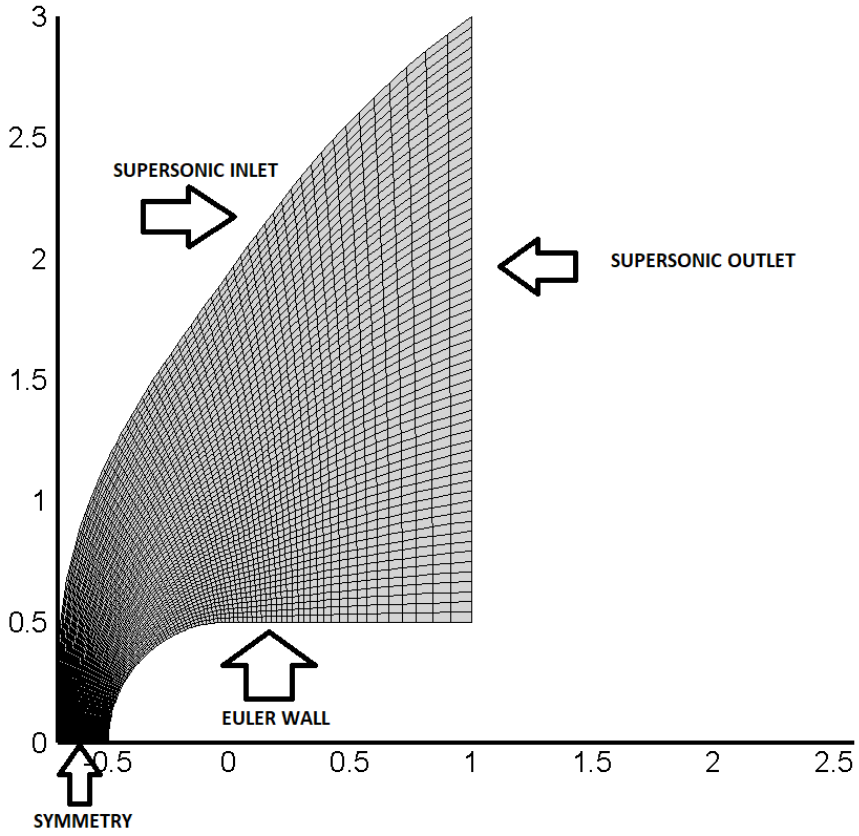


Figure 5.33: Mesh and boundary conditions for supersonic flow over blunt body

As free-stream conditions we pick Mach number $Ma_\infty = 12$, pressure $p_\infty = 43\,\text{Pa}$ and temperature $T_\infty = 266\,\text{K}$.
The chosen composition is 79% of N_2 and 21% of O_2, while the chemical scheme from [35] is the following:

$$1.\ O_2 + N \longleftrightarrow 2O + N$$

$$2.\ O_2 + NO \longleftrightarrow 2O + NO$$

$$3.\ O_2 + O \longleftrightarrow 2O + O$$

$$4.\ O_2 + O_2 \longleftrightarrow 2O + O_2$$

$$5.\ O_2 + N_2 \longleftrightarrow 2O + N_2$$

$$6.\ N_2 + O \longleftrightarrow 2N + O$$

7. $N_2 + NO \longleftrightarrow 2N + NO$

8. $N_2 + O_2 \longleftrightarrow 2N + O_2$

9. $N_2 + N \longleftrightarrow 2N + N$

10. $N_2 + N_2 \longleftrightarrow 2N + N_2$

11. $NO + O_2 \longleftrightarrow N + O + O_2$

12. $NO + N_2 \longleftrightarrow N + O + N_2$

13. $NO + O \longleftrightarrow N + O + O$

14. $NO + N \longleftrightarrow N + O + N$

15. $NO + NO \longleftrightarrow N + O + NO$

16. $NO + O \longleftrightarrow O_2 + N$

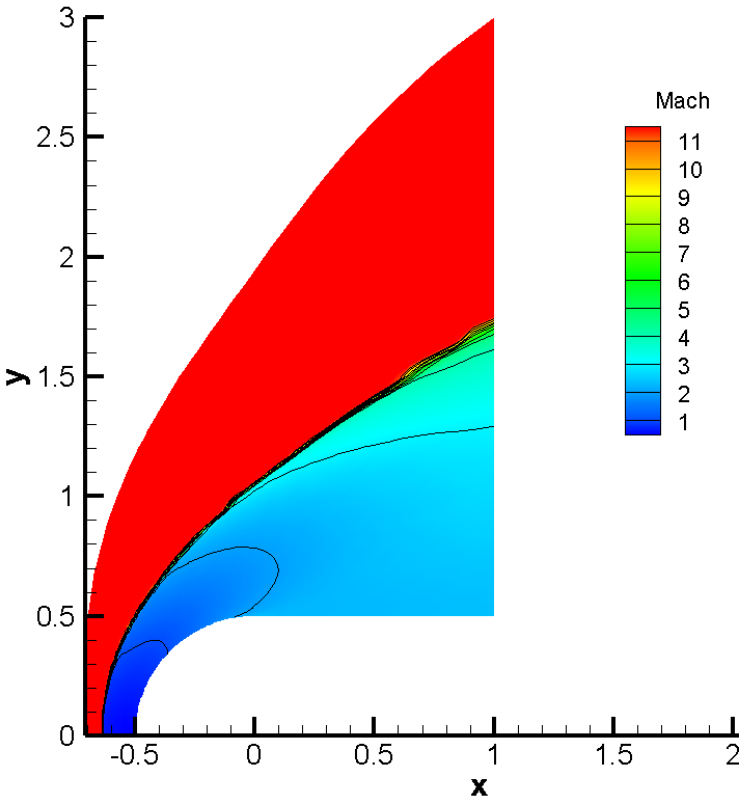
17. $N_2 + 0 \longleftrightarrow NO + N$

with the following parameters

N. reaction	Arrhenius parameter	Temperature exponent	Activation temperature
1	3.610e+18	-1	5.940e+04
2	3.610e+18	-1	5.940e+04
3	3.610e+18	-1	5.940e+04
4	3.610e+19	-1	5.940e+04
5	3.610e+18	-1	5.940e+04
6	1.920e+17	-0.5	1.131e+05
7	1.920e+17	-0.5	1.131e+05
8	1.920e+17	-0.5	1.131e+05
9	4.150e+22	-1.5	1.131e+05
10	1.920e+17	-0.5	1.131e+05
11	3.970e+20	-1.5	7.560e+04
12	3.970e+20	-1.5	7.560e+04
13	3.970e+20	-1.5	7.560e+04
14	3.970e+20	-1.5	7.560e+04
15	3.970e+20	-1.5	7.560e+04
16	3.180e+09	1	1.970e+04
17	6.750e+13	0	3.750e+04

Table 5.3: Parameters of Gupta [35] chemical scheme: units are $\mathrm{cm}, \mathrm{mol}, \mathrm{s}, \mathrm{K}$

It is worth to notice that the exponent coefficients to compute reaction rates coincide with stoichiometric coefficients.
Moreover we specify that data for thermodynamic properties are taken by [39].
Let us report now the obtained results:



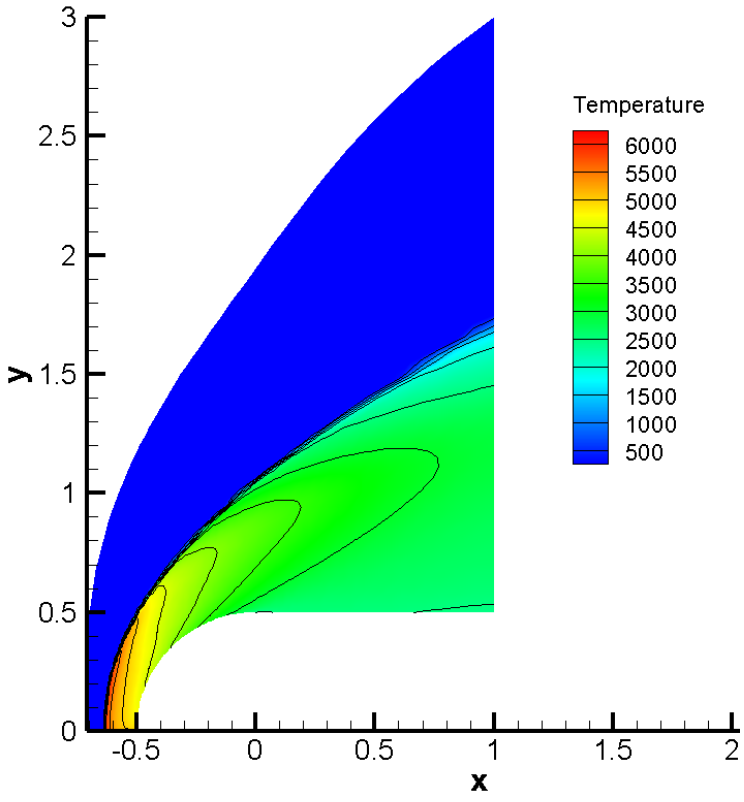
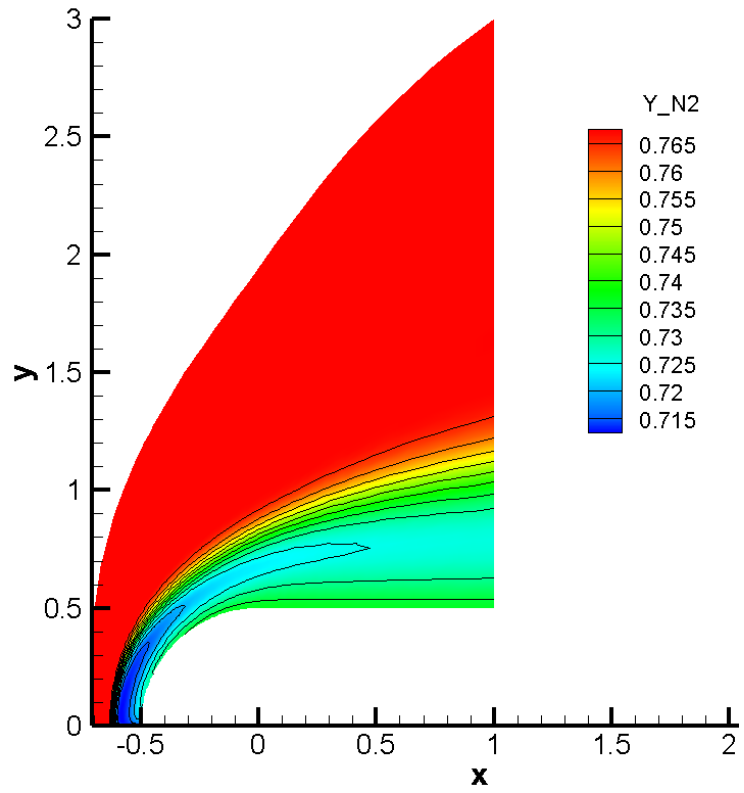


Figure 5.34: Contour plot of Mach number (left) and temperature (right)



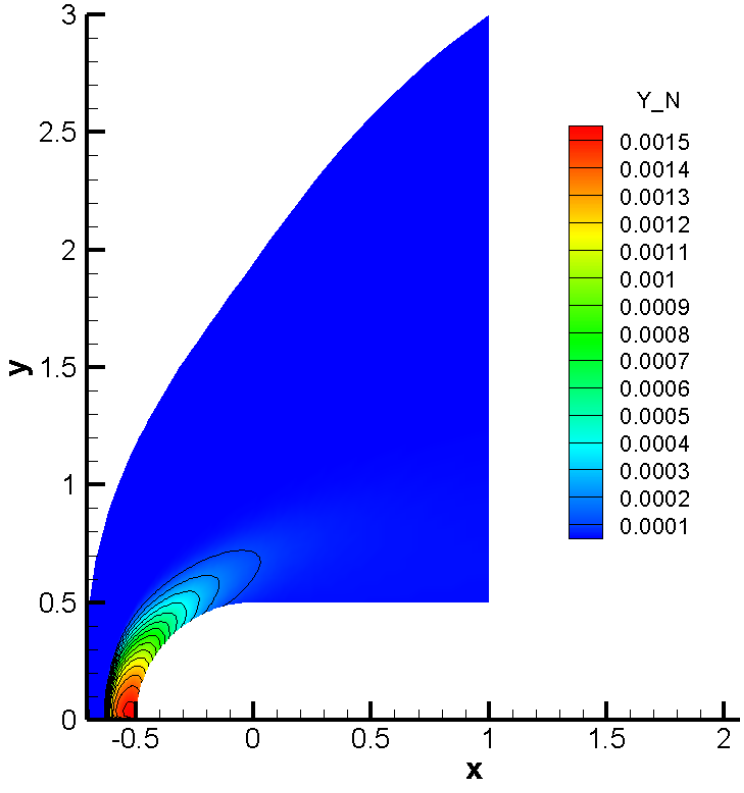


Figure 5.35: Distribution of N_2 mass fraction (left) and N mass fraction (right)

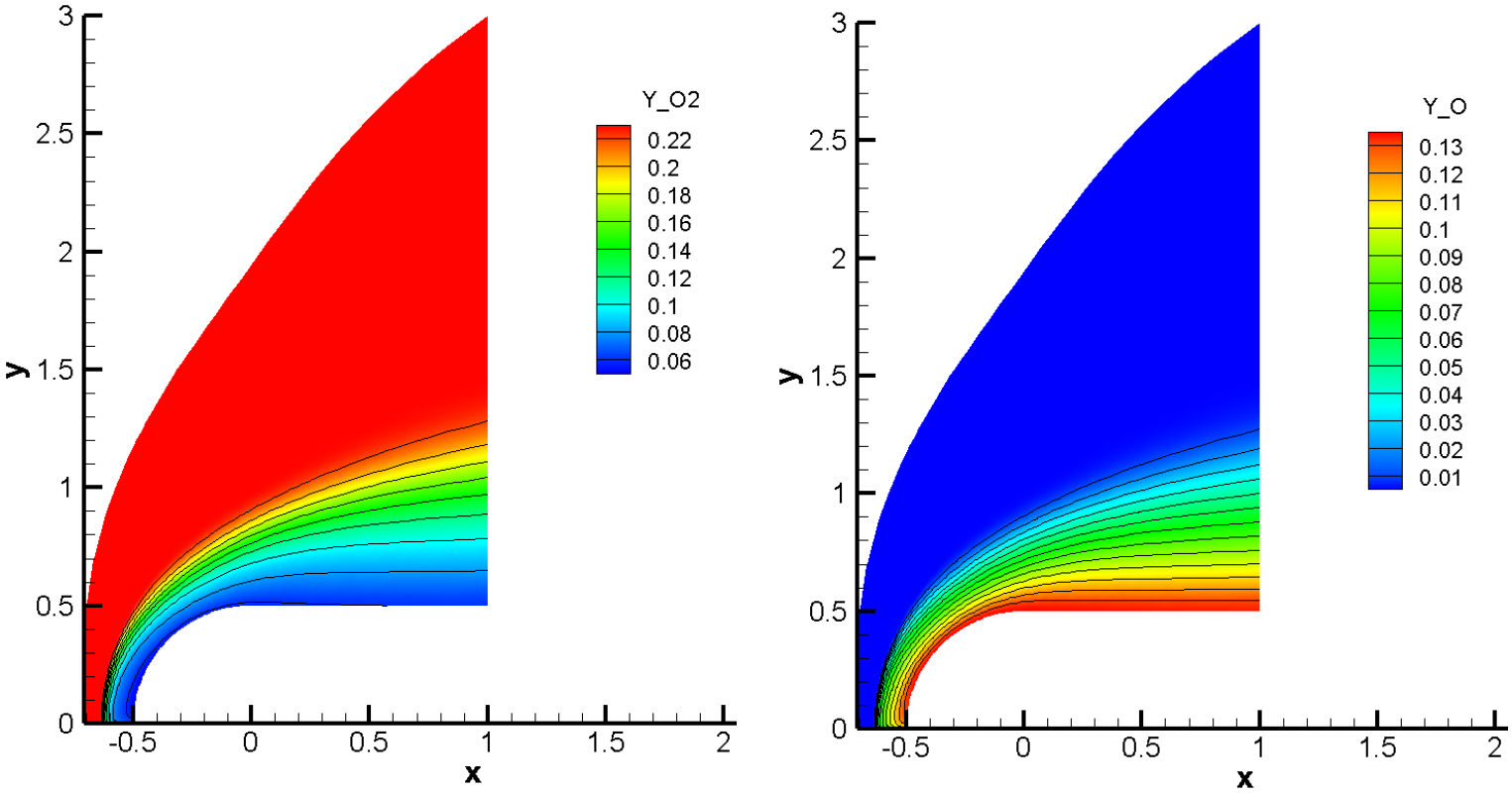


Figure 5.36: Distribution of O_2 mass fraction (left) and O mass fraction (right)

As we can notice from Mach number and temperature contour plots we see the shock wave who is travelling along the domain.
For what concerns the mixture composition we underline the fact that the nitrogen N_2 dissociates into N minimally, while the molecular oxygen O_2 dissociates into atomic oxygen in significant quantity.

Moreover we report the contour plot of the total enthalpy which is conserved along the domain as expected since we are using Euler equations

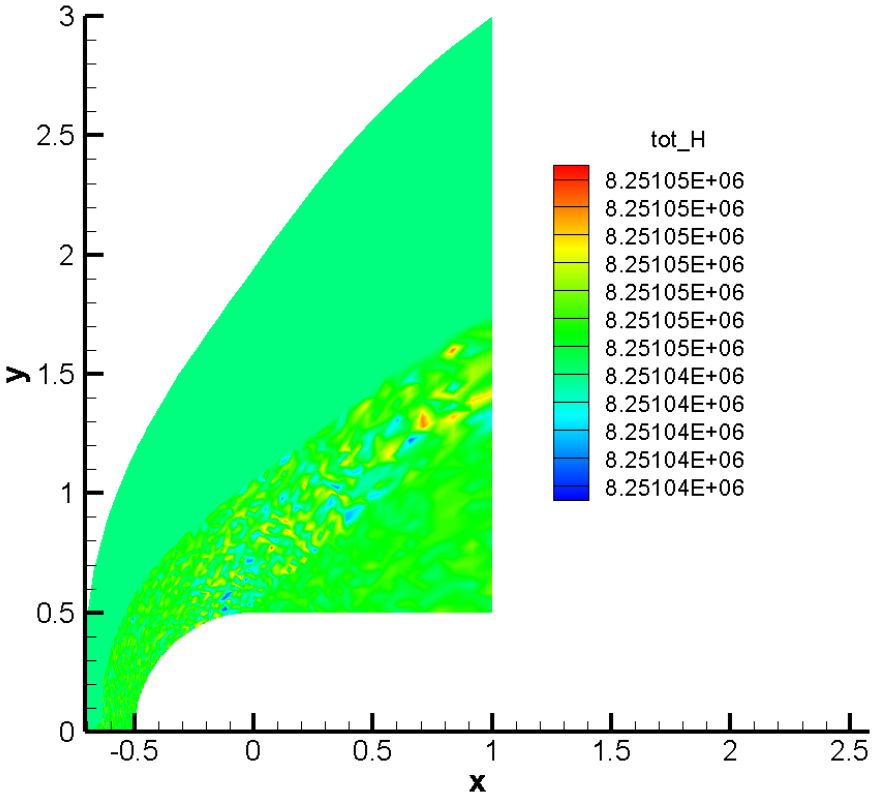


Figure 5.37: Contour plot of total enthalpy

Then we tried to perform a simulation using a 2nd order scheme. Unfortunately in this case the obtained results are not satisfactory

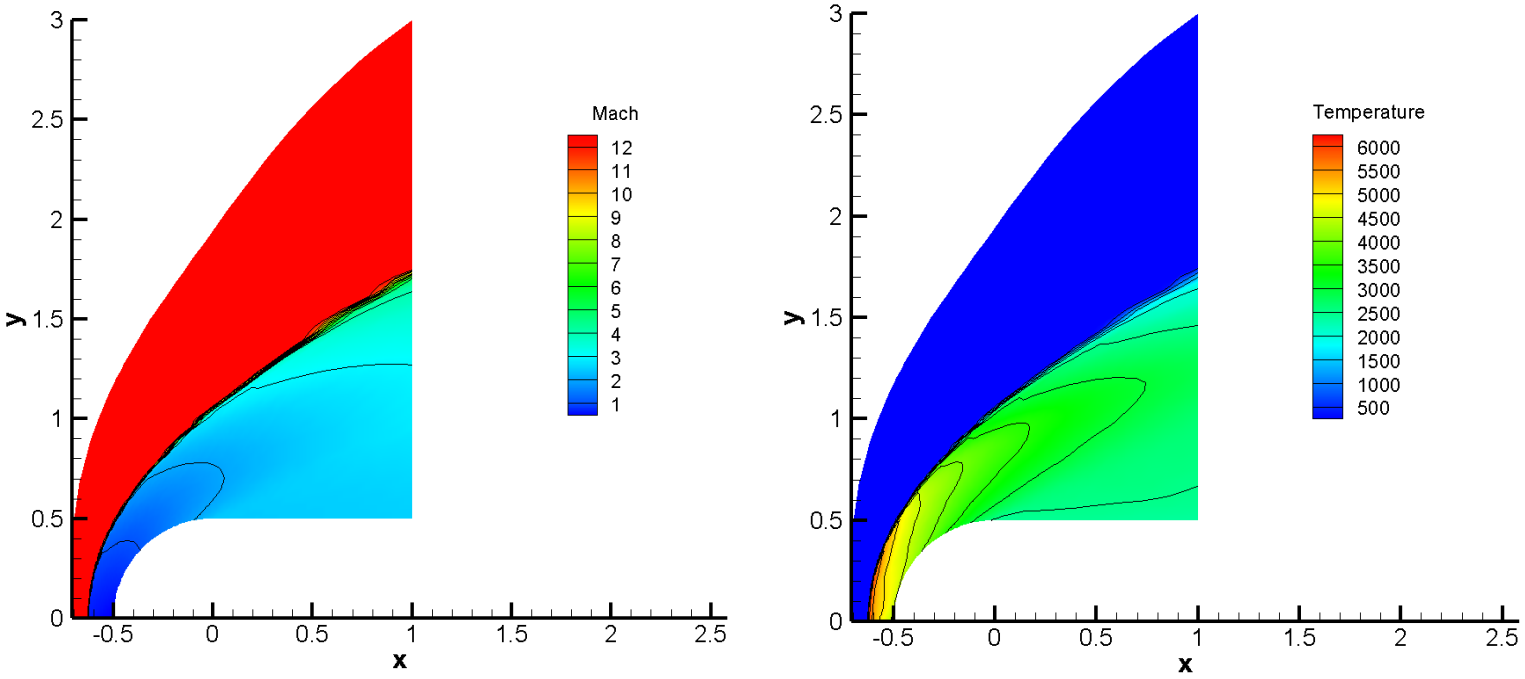


Figure 5.38: Contour plot of Mach number (left) and temperature (right) with a second order scheme

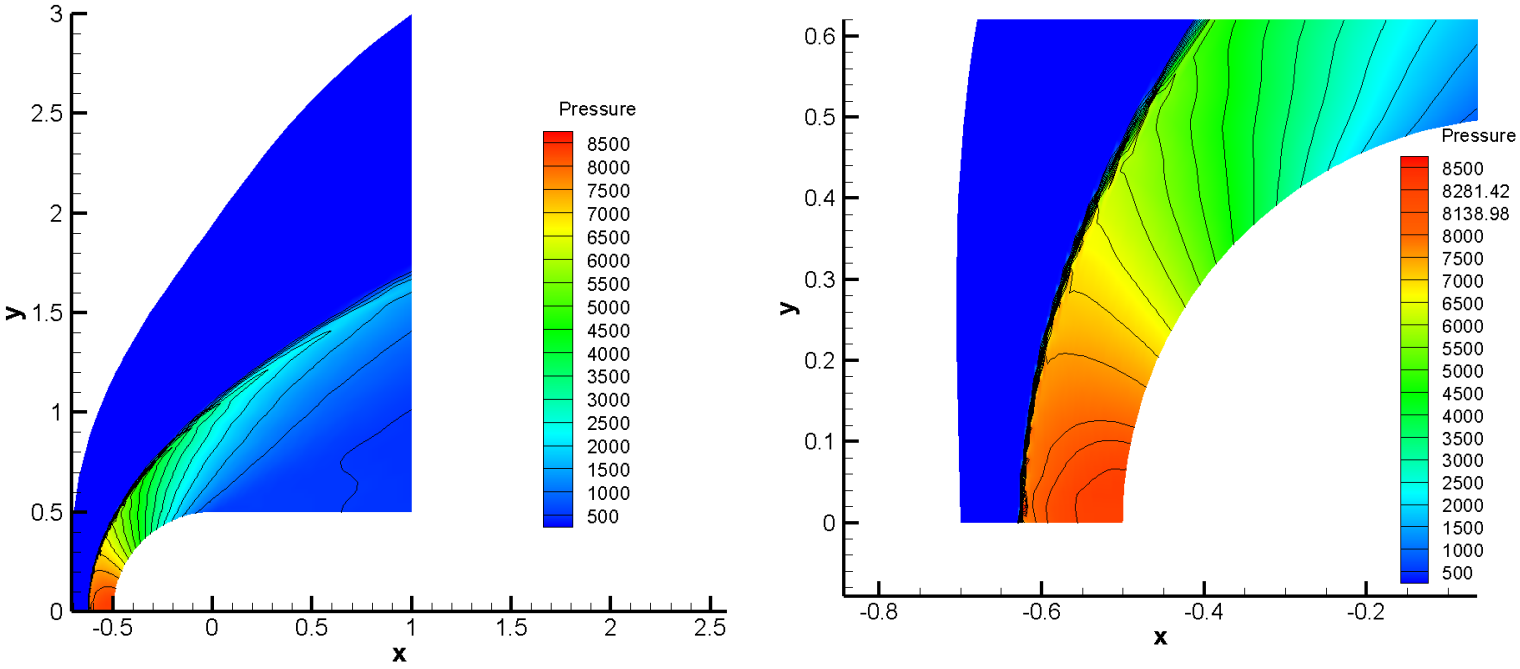


Figure 5.39: Contour plot of pressure (left) and relative zoom (right) with a second order scheme

We do not retrieve the expected accuracy as it can be noticed by the oscillations present in the results.
Moreover there is another issue: the total enthalpy is no more conserved; even if we cannot expect the perfect results obtained in Figure (5.37) because of the reconstruction procedure, we find large oscillations, while in [38] it has been shown that the relative difference is on the order of 0.01%.

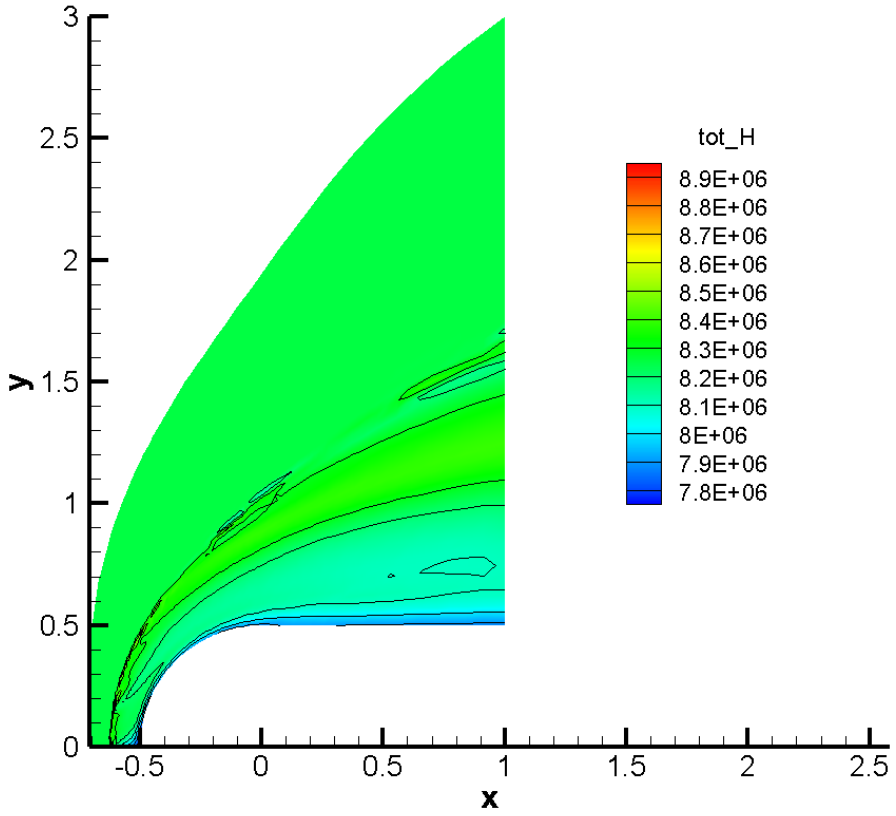


Figure 5.40: Contour plot of total enthalpy with a second order scheme

Since we saw through the simulation with a first order scheme that there are no problems in imposing the boundary conditions, the reason of this issue is probably due to the reconstruction procedure and in particular to the limiter.

Chapter 6

Conclusions

In this work we developed a numerical solver for the simulation of reacting flows with a novel approach in the treatment and computation of molecular diffusion fluxes which play an essential role in the progress of chemical reactions.
Different numerical simulations were performed in order to validate the code and they showed a good behaviour both for subsonic and supersonic flows.
It is worth to notice that the code is able to support 3D simulations in case more complex geometries are needed.

Future developments and extensions are possible, for instance:

1. The capability to treat turbulent flows;
2. The implementation of adequate reconstruction schemes to simulate correctly hypersonic flows;
3. The implementation of catalytic boundary condition;
4. The implementation of a radiation model, able to describe precisely the radiative heating effects inside a combustion chamber.

The first suggestion requires further discussion: usually the closure of governing equations for turbulent flows is based on strong approximations and hypothesis. In his work, Mazzetti [1] found an accurate closure for the turbulent transport of energy, but the same does not hold for diffusion fluxes where a lot of work have still to be done specially in case Stefan-Maxwell equations are employed: it is certainly a difficult issue.

Appendix A

A.1 Equations formulated for moving grids

Many unsteady flows require solutions on arbitrary moving grids; for this reason we report here a popular approach to analyse this kind of situations known as *Arbitrary Lagrangian Eulerian* (ALE) formulation.
This method leads to a modified form of the Navier-Stokes equations which accounts for the relative motion of the grid with respect to the fluid.

The governing equations (2.17) written in a time-dependent integral form for a moving control volume Ω with boundary Σ, surface element $d\Sigma$ and $\boldsymbol{n}$ as outward unit normal vector of the boundary Σ read as follows:

$$\int_\Omega \frac{\partial \boldsymbol{Q}}{\partial t} d\Omega + \int_\Sigma \boldsymbol{F^M n} d\Sigma - \int_\Sigma \boldsymbol{G n} d\Sigma - \int_\Omega \boldsymbol{S} d\Omega = 0 \tag{A.1}$$

The only difference with respect to the integral form (3.2) is in the definition of $\boldsymbol{F^M}$, while the vector of viscous fluxes $\boldsymbol{G}$, the source term $\boldsymbol{Q}$ and the components of the stress tensor $\boldsymbol{\tau}$ retain the same form.

The vector of convective fluxes projected onto $\boldsymbol{n}$ becomes on dynamic grids

$$\boldsymbol{F^M n} = \boldsymbol{F n} - (\boldsymbol{v_{loc}} \cdot \boldsymbol{n}) \boldsymbol{Q}$$

where

$$\boldsymbol{v_{loc}} = \left(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}, \frac{\partial z}{\partial t} \right)^T$$

is the local velocity for a domain in motion.

Summing up we can express the new term $\boldsymbol{F^M}$ as:

$$\boldsymbol{F^M} = \begin{pmatrix} \rho(\boldsymbol{u} - \boldsymbol{v_{loc}}) \\ \rho \boldsymbol{u} \otimes (\boldsymbol{u} - \boldsymbol{v_{loc}}) + p\boldsymbol{I} \\ \rho E(\boldsymbol{u} - \boldsymbol{v_{loc}}) + p\boldsymbol{u} \\ \rho_1(\boldsymbol{u} - \boldsymbol{v_{loc}}) \\ \rho_2(\boldsymbol{u} - \boldsymbol{v_{loc}}) \\ ... \\ \rho_{N_S}(\boldsymbol{u} - \boldsymbol{v_{loc}}) \end{pmatrix} = \begin{pmatrix} \rho(\boldsymbol{u} - \boldsymbol{v_{loc}}) \\ \rho \boldsymbol{u} \otimes (\boldsymbol{u} - \boldsymbol{v_{loc}}) + p\boldsymbol{I} \\ (\rho E + p)(\boldsymbol{u} - \boldsymbol{v_{loc}}) + p\boldsymbol{v_{loc}} \\ \rho_1(\boldsymbol{u} - \boldsymbol{v_{loc}}) \\ \rho_2(\boldsymbol{u} - \boldsymbol{v_{loc}}) \\ ... \\ \rho_{N_S}(\boldsymbol{u} - \boldsymbol{v_{loc}}) \end{pmatrix}$$

The mass flux $\dot{m}$ and the vector $\mathbf{\Psi}$ of the AUSM method reported in (3.6) become:

$$\dot{m} = \rho(\boldsymbol{u} - \boldsymbol{v_{loc}}) \cdot \boldsymbol{n}, \mathbf{\Psi} = \begin{pmatrix} 1 \\ \boldsymbol{u} \\ H + \frac{p\boldsymbol{v_{loc}} \cdot \boldsymbol{n}}{\rho(\boldsymbol{u} - \boldsymbol{v_{loc}}) \cdot \boldsymbol{n}} \\ Y_i \end{pmatrix}$$

Moreover the modified flow tangency wall boundary condition is $(\boldsymbol{u} - \boldsymbol{v_{loc}}) \cdot \boldsymbol{n} = 0$ for inviscid flows and the no-slip wall boundary condition becomes $\boldsymbol{u} = \boldsymbol{v_{loc}}$.

As reported in [27], Thomas and Lombard pointed out first that besides the conservation of mass, momentum and energy, also the so-called *Geometric Conservation Law*(GCL) must be satisfied in order to avoid errors induced by moving control volumes; the integral form of GCL reads:

$$\frac{\partial}{\partial t} \int_{\Omega} d\Omega - \int_{\Sigma} \boldsymbol{v_{loc}} \cdot \boldsymbol{n} d\Sigma = 0$$

and this result comes from the requirement that a numerical scheme employed in the simulations must be able to reproduce a constant solution (uniform flow) independently on the deformation of the grid.

A.2 Pressure and Temperature Derivatives

Let us recall the employed state equation for a mixture that obeys to the perfect gas relation:

$$p = \sum_{i=1}^{N_S} \rho_i R_i T \tag{A.2}$$

As we can see from (A.2) the thermodynamic quantity p is described by $N_S + 1$ variables and therefore $p = p(\rho_i, T)$.
Moreover since T is not an unknown of the Navier-Stokes equations it must exist and additional relation between the temperature and the conserved variables $T(\boldsymbol{Q})$ so that $p = p(\rho_i, T(\boldsymbol{Q})) = p(\boldsymbol{Q})$.

In order to express the Jacobian matrices for convective and viscous fluxes we are interested in the derivatives of pressure and temperature with respect to $\boldsymbol{Q}$.
Let us start from (A.2):

$$dp = \sum_{i=1}^{N_S} R_i T d\rho_i + \sum_{i=1}^{N_S} \rho_i R_i dT = \sum_{i=1}^{N_S} R_i T d\rho_i + \rho R dT \tag{A.3}$$

At this point we need to know dT in order to proceed; for this purpose we refer to another thermodynamic quantity: the internal energy. Indeed we get:

$$\rho e = \sum_{i=1}^{N_S} \rho_i e_i \implies d(\rho e) = \sum_{i=1}^{N_S} e_i d\rho_i + \sum_{i=1}^{N_S} \rho_i de_i \tag{A.4}$$

Since each gas is considered thermally perfect, the internal energy of each species i is a function of temperature only so that $de_i = \frac{de_i}{dT} dT = C_{v_i} dT$; if we substitute into (A.4) we obtain:

$$\begin{aligned} d(\rho e) &= \sum_{i=1}^{N_S} e_i d\rho_i + \sum_{i=1}^{N_S} \rho_i C_{v_i} dT = \sum_{i=1}^{N_S} e_i d\rho_i + \rho C_v dT \\ \implies dT &= \frac{d(\rho e) - \sum_{i=1}^{N_S} e_i d\rho_i}{\rho C_v} \end{aligned} \tag{A.5}$$

Now we need an expression for $d(\rho e)$ in terms of the unknowns $\boldsymbol{Q}$ and since the most related conserved variable to the internal energy is ρE we try to employ the following relation:

$$\rho E = \rho e + \frac{1}{2}\rho \boldsymbol{u} \cdot \boldsymbol{u} = \rho e + \frac{1}{2}\frac{(\rho \boldsymbol{u}) \cdot (\rho \boldsymbol{u})}{\rho} \implies d(\rho E) = d(\rho e) + \frac{1}{2} d\left(\frac{(\rho \boldsymbol{u}) \cdot (\rho \boldsymbol{u})}{\rho}\right)$$

The term $d\left(\frac{(\rho \boldsymbol{u}) \cdot (\rho \boldsymbol{u})}{\rho}\right)$ can be expanded in this way:

$$d\left(\frac{(\rho \boldsymbol{u}) \cdot (\rho \boldsymbol{u})}{\rho}\right) = 2\frac{(\rho \boldsymbol{u}) \cdot d(\rho \boldsymbol{u})}{\rho} + (\rho \boldsymbol{u}) \cdot (\rho \boldsymbol{u}) d(\frac{1}{\rho}) = 2\boldsymbol{u} \cdot d(\rho \boldsymbol{u}) - (\boldsymbol{u} \cdot \boldsymbol{u}) d\rho$$

Therefore we finally obtain:

$$d(\rho E) = d(\rho e) + \boldsymbol{u} \cdot d(\rho \boldsymbol{u}) - \frac{1}{2}(\boldsymbol{u} \cdot \boldsymbol{u}) d\rho \implies d(\rho e) = \frac{1}{2}(\boldsymbol{u} \cdot \boldsymbol{u}) d\rho - \boldsymbol{u} \cdot d(\rho \boldsymbol{u}) + d(\rho E) \tag{A.6}$$

Now we can back substitute (A.6) into (A.5) to find:

$$dT = \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2\rho C_v}d\rho - \frac{\boldsymbol{u}}{\rho C_v}\cdot d(\rho\boldsymbol{u}) + \frac{1}{\rho C_v}d(\rho E) - \sum_{i=1}^{N_S}\frac{e_i}{\rho C_v}d\rho_i \tag{A.7}$$

Eventually we need to back substitute (A.7) into (A.3) to conclude the computations related to pressure derivatives:

$$\begin{aligned} dp &= \sum_{i=1}^{N_S} R_i T d\rho_i + \rho R\left(\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2\rho C_v}d\rho - \frac{\boldsymbol{u}}{\rho C_v}\cdot d(\rho\boldsymbol{u}) + \frac{1}{\rho C_v}d(\rho E) - \sum_{i=1}^{N_S}\frac{e_i}{\rho C_v}d\rho_i\right) \\ &= \frac{R}{C_v}\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2}d\rho - \frac{R}{C_v}\boldsymbol{u}\cdot d(\rho\boldsymbol{u}) + \frac{R}{C_v}d(\rho E) + \sum_{i=1}^{N_S}\left(R_iT - \frac{R}{C_v}e_i\right)d\rho_i \end{aligned} \tag{A.8}$$

Moreover the following relation holds:

$$\frac{R}{C_v} = \frac{C_p - C_v}{C_v} = \gamma - 1 \tag{A.9}$$

Summing up, thanks to (A.7),(A.8) and (A.9) we found:

$$\frac{\partial T(\boldsymbol{Q})}{\partial \rho} = \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2\rho C_v} \qquad \frac{\partial T(\boldsymbol{Q})}{\partial(\rho\boldsymbol{u})} = -\frac{\boldsymbol{u}}{\rho C_v} \tag{A.10}$$

$$\frac{\partial T(\boldsymbol{Q})}{\partial(\rho E)} = \frac{1}{\rho C_v} \qquad \frac{\partial T(\boldsymbol{Q})}{\partial \rho_i} = -\frac{e_i}{\rho C_v} \tag{A.11}$$

(A.12)

$$\frac{\partial p(\boldsymbol{Q})}{\partial \rho} = (\gamma-1)\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2} \qquad \frac{\partial p(\boldsymbol{Q})}{\partial(\rho\boldsymbol{u})} = -(\gamma-1)\boldsymbol{u} \tag{A.13}$$

$$\frac{\partial p(\boldsymbol{Q})}{\partial(\rho E)} = (\gamma-1) \qquad \frac{\partial p(\boldsymbol{Q})}{\partial \rho_i} = R_iT - (\gamma-1)e_i \tag{A.14}$$

A.3 Convective Flux Jacobian

The convective flux Jacobian represents the gradient of the convective fluxes with respect to the conservative variables.
First of all let us report the definition of the aforementioned fluxes in a more convenient form:

$$\boldsymbol{F}\boldsymbol{n} = \boldsymbol{F_n} = \begin{pmatrix} \rho\boldsymbol{u}\cdot\boldsymbol{n} \\ (\rho\boldsymbol{u}\otimes\boldsymbol{u} + p\boldsymbol{I})\,\boldsymbol{n} \\ (\rho E + p)\boldsymbol{u}\cdot\boldsymbol{n} \\ \rho_1\boldsymbol{u}\cdot\boldsymbol{n} \\ \rho_2\boldsymbol{u}\cdot\boldsymbol{n} \\ ... \\ \rho_{N_S}\boldsymbol{u}\cdot\boldsymbol{n} \end{pmatrix} = \begin{pmatrix} \rho\boldsymbol{u}\cdot\boldsymbol{n} \\ \frac{(\rho\boldsymbol{u}\otimes\rho\boldsymbol{u})}{\rho}\boldsymbol{n} + p\boldsymbol{n} \\ (\rho E + p)\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho} \\ \rho_1\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho} \\ \rho_2\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho} \\ ... \\ \rho_{N_S}\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho} \end{pmatrix}$$

For simplicity let us denote by F_n^1 the term $\rho\boldsymbol{u}\cdot\boldsymbol{n}$, by $\boldsymbol{F_n^2}$ the term $\frac{(\rho\boldsymbol{u}\otimes\rho\boldsymbol{u})}{\rho}\boldsymbol{n} + p\boldsymbol{n}$, by F_n^3 the term $(\rho E + p)\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho}$ and by F_n^4 the term $\rho_i\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho}$.
For what concerns F_n^1 we get:

$$\frac{\partial F_n^1}{\partial\rho} = 0 \qquad \frac{\partial F_n^1}{\partial(\rho\boldsymbol{u})} = \boldsymbol{n} \qquad \frac{\partial F_n^1}{\partial\rho E} = 0 \qquad \frac{\partial F_n^1}{\partial\rho_i} = 0 \tag{A.15}$$

About $\boldsymbol{F_n^2}$ we will find the derivatives only along x direction and the ones along y and z will follow immediately from similar computations. Let $F_{n_x}^2 = \rho u_x\frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho} + pn_x$ the component of F_n^2 along x; hence it follows:

$$\begin{aligned}
&\frac{\partial F_{n_x}^2}{\partial\rho} = -\frac{1}{\rho^2}\rho u_x(\rho\boldsymbol{u}\cdot\boldsymbol{n}) + \frac{\partial p}{\partial\rho}n_x = -u_x\boldsymbol{u}\cdot\boldsymbol{n} + \frac{\partial p}{\partial\rho}n_x \\
&\frac{\partial F_{n_x}^2}{\partial(\rho u_x)} = \frac{\rho\boldsymbol{u}\cdot\boldsymbol{n}}{\rho} + \frac{\rho u_x n_x}{\rho} + \frac{\partial p}{\partial\rho u_x}n_x = \boldsymbol{u}\cdot\boldsymbol{n} + u_x n_x + \frac{\partial p}{\partial\rho u_x}n_x \\
&\frac{\partial F_{n_x}^2}{\partial(\rho u_y)} = \frac{\rho u_x n_y}{\rho} + \frac{\partial p}{\partial\rho u_y}n_x = u_x n_y + \frac{\partial p}{\partial\rho u_y}n_x \qquad \frac{\partial F_{n_x}^2}{\partial(\rho u_z)} = \frac{\rho u_x n_z}{\rho} + \frac{\partial p}{\partial\rho u_z}n_x = u_x n_z + \frac{\partial p}{\partial\rho u_z}n_x \\
&\frac{\partial F_{n_x}^2}{\partial\rho E} = \frac{\partial p}{\partial\rho E}n_x \qquad \frac{\partial F_{n_x}^2}{\partial\rho_i} = \frac{\partial p}{\partial\rho_i}n_x
\end{aligned} \tag{A.16}$$

Summing up we obtain:

$$\begin{aligned}
&\frac{\partial\boldsymbol{F_n^2}}{\partial\rho} = -\boldsymbol{u}(\boldsymbol{u}\cdot\boldsymbol{n}) + \frac{\partial p}{\partial\rho}\boldsymbol{n} \qquad \frac{\partial F_{n_i}^2}{\partial(\rho u)_j} = \boldsymbol{u}\cdot\boldsymbol{n}\delta_{ij} + u_i n_j + \frac{\partial p}{(\partial\rho u)_j}n_i, \qquad i,j = 1...3 \\
&\frac{\partial\boldsymbol{F_n^2}}{\partial\rho E} = \frac{\partial p}{\partial\rho E}\boldsymbol{n} \qquad \frac{\partial\boldsymbol{F_n^2}}{\partial\rho_i} = \frac{\partial p}{\partial\rho_i}\boldsymbol{n}
\end{aligned} \tag{A.17}$$

For what concerns F_n^3 we find:

$$\begin{aligned}
&\frac{\partial F_n^3}{\partial\rho} = -\frac{1}{\rho^2}((\rho\boldsymbol{u}\cdot\boldsymbol{n})\rho E) + \rho\boldsymbol{u}\cdot\boldsymbol{n}\frac{\partial\left(\frac{p}{\rho}\right)}{\partial\rho} = -E(\boldsymbol{u}\cdot\boldsymbol{n}) + \frac{\partial p}{\partial\rho}(\boldsymbol{u}\cdot\boldsymbol{n}) - \frac{p}{\rho}(\boldsymbol{u}\cdot\boldsymbol{n}) = (\boldsymbol{u}\cdot\boldsymbol{n})\left(\frac{\partial p}{\partial\rho} - E - \frac{p}{\rho}\right) \\
&\frac{\partial F_n^3}{\partial(\rho\boldsymbol{u})} = \frac{\rho E}{\rho}\boldsymbol{n} + \frac{p}{\rho}\boldsymbol{n} + (\boldsymbol{u}\cdot\boldsymbol{n})\frac{\partial p}{\partial(\rho\boldsymbol{u})} = \left(E + \frac{p}{\rho}\right)\boldsymbol{n} + \frac{\partial p}{\partial(\rho\boldsymbol{u})}(\boldsymbol{u}\cdot\boldsymbol{n}) \\
&\frac{\partial F_n^3}{\partial\rho E} = \boldsymbol{u}\cdot\boldsymbol{n} + (\boldsymbol{u}\cdot\boldsymbol{n})\frac{\partial p}{\partial(\rho E)} = \left(\frac{\partial p}{\partial(\rho E)} + 1\right)(\boldsymbol{u}\cdot\boldsymbol{n}) = \gamma(\boldsymbol{u}\cdot\boldsymbol{n}) \\
&\frac{\partial F_n^3}{\partial\rho_j} = (\boldsymbol{u}\cdot\boldsymbol{n})\frac{\partial p}{\partial\rho_j}
\end{aligned} \tag{A.18}$$

Let us focus finally on F_n^4:

$$\frac{\partial F_n^4}{\partial \rho} = -\frac{\rho_i}{\rho^2}(\rho \boldsymbol{u} \cdot \boldsymbol{n}) = -Y_i\,(\boldsymbol{u} \cdot \boldsymbol{n}) \qquad \frac{\partial F_n^4}{\partial (\rho \boldsymbol{u})} = Y_i \boldsymbol{n}$$
$$\frac{\partial F_n^4}{\partial \rho E} = 0 \qquad \frac{\partial F_n^4}{\partial \rho_j} = (\boldsymbol{u} \cdot \boldsymbol{n})\delta_{ij} \tag{A.19}$$

A.4 AUSM Scheme Approximate Jacobian

The numerical flux computed by means of AUSM scheme reported in Section (3.1.2) can be expressed as a function of the conserved variables as:

$$\begin{aligned}\boldsymbol{F}_{1/2}\left(\boldsymbol{Q}_L,\boldsymbol{Q}_R,\boldsymbol{n}\right) =& a_{1/2}\left\{ M^+_{1/2}\boldsymbol{Q}_L + M^+_{1/2}\begin{pmatrix}0\\ \boldsymbol{0}\\ p_L\\ 0\\ 0\\ ...\\ 0\end{pmatrix} + M^-_{1/2}\boldsymbol{Q}_R + M^-_{1/2}\begin{pmatrix}0\\ \boldsymbol{0}\\ p_R\\ 0\\ 0\\ ...\\ 0\end{pmatrix}\right\}\\ &+ p_{1/2}\begin{pmatrix}0\\ \boldsymbol{n}\\ 0\\ 0\\ 0\\ ...\\ 0\end{pmatrix}\end{aligned} \tag{A.20}$$

Herein $M^{\pm}_{1/2} = \frac{1}{2}\left(M_{1/2} + \left|M_{1/2}\right|\right)$ with

$$M_{1/2} = \mathcal{M}^+_{(4)}(M_L) + \mathcal{M}^-_{(4)}(M_R) - M_P(M_L, M_R)$$

where

$$M_P(M_L, M_R) = \frac{K_p}{f_a}\max\left(0, 1-\sigma\bar{M}^2\right)\frac{p_R - p_L}{\rho_{1/2}a^2_{1/2}}$$

and

$$p_{1/2} = \mathcal{P}^+(M_L)p_L + \mathcal{P}^-(M_R)p_R - P_P(M_L, M_R)$$

with

$$P_P(M_L, M_R) = K_u\mathcal{P}^+_{(5)}(M_L)\mathcal{P}^-_{(5)}(M_R)\left(\rho_L + \rho_R\right)\left(f_a a_{1/2}\right)\left(\boldsymbol{u}_R\cdot\boldsymbol{n} - \boldsymbol{u}_L\cdot\boldsymbol{n}\right)$$

All the other terms are defined in Section (3.1.2).

Since we are interested in an approximate Jacobian, we will neglect the dependence of the interface speed of sound on the conserved variables (frozen speed of sound); hence

the complete Jacobian of the scheme can be written in compact form as:

$$\frac{\partial \boldsymbol{F}_{1/2}}{\partial \boldsymbol{Q}_{L/R}} = a_{1/2} \begin{pmatrix} \left(\frac{\partial M^+_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_L + \left(\frac{\partial M^-_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_R \\ \left(\frac{\partial M^+_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T (\rho \boldsymbol{u})_L + \left(\frac{\partial M^-_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T (\rho \boldsymbol{u})_R \\ \left(\frac{\partial M^+_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T ((\rho E)_L + p_L) + M^{+,-}_{1/2} \left(\frac{\partial p_{L,R}}{\partial \boldsymbol{Q}_{L,R}}\right)^T + \left(\frac{\partial M^-_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T ((\rho E)_R + p_R) \\ \left(\frac{\partial M^+_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_{1_L} + \left(\frac{\partial M^-_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_{1_R} \\ \left(\frac{\partial M^+_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_{2_L} + \left(\frac{\partial M^-_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_{2_R} \\ \dots \\ \left(\frac{\partial M^+_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_{N_{S_L}} + \left(\frac{\partial M^-_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \rho_{N_{S_R}} \end{pmatrix} + a_{1/2} M^{+,-}_{1/2} \boldsymbol{I} + \begin{pmatrix} \boldsymbol{0}^T \\ \left(\frac{\partial p_{1/2}}{\partial \boldsymbol{Q}_{L/R}}\right)^T \boldsymbol{n} \\ \boldsymbol{0}^T \\ \boldsymbol{0}^T \\ \boldsymbol{0}^T \\ \dots \\ \boldsymbol{0}^T \end{pmatrix} \tag{A.21}$$

Now we can focus on the expressions of the partial derivatives in (A.21):

$$\frac{\partial M^{\pm}_{1/2}}{\partial \boldsymbol{Q}_L} = \frac{1}{2} \frac{\partial M_{1/2}}{\partial \boldsymbol{Q}_L} \left(1 \pm \text{sgn}(M_{1/2})\right) \qquad \frac{\partial M^{\pm}_{1/2}}{\partial \boldsymbol{Q}_R} = \frac{1}{2} \frac{\partial M_{1/2}}{\partial \boldsymbol{Q}_R} \left(1 \pm \text{sgn}(M_{1/2})\right) \tag{A.22}$$

$$\frac{\partial M_{1/2}}{\partial \boldsymbol{Q}_L} = \frac{\partial \mathcal{M}^+_{(4)}}{\partial \boldsymbol{Q}_L} - \frac{\partial M_P}{\partial \boldsymbol{Q}_L} \qquad \frac{\partial M_{1/2}}{\partial \boldsymbol{Q}_R} = \frac{\partial \mathcal{M}^-_{(4)}}{\partial \boldsymbol{Q}_R} - \frac{\partial M_P}{\partial \boldsymbol{Q}_R} \tag{A.23}$$

$$\frac{\partial \mathcal{M}^{\pm}_{(4)}(M)}{\partial \boldsymbol{Q}} = \begin{cases} \frac{1}{2}\left(1 \pm \text{sgn}(M)\right) \frac{\partial M}{\partial \boldsymbol{Q}}, & \text{if } |M| \geq 1 \\ \pm\left(\frac{1}{2} \pm 1\right)\left(1 + 8\beta M\left(M \mp 1\right)\right) \frac{\partial M}{\partial \boldsymbol{Q}}, & \text{otherwise} \end{cases} \tag{A.24}$$

$$\frac{\partial M_P(M_L, M_R)}{\partial \boldsymbol{Q_L}} = \frac{K_p}{a^2_{1/2}} \frac{\partial \left(\frac{\max(0, 1-\sigma \bar{M}^2)}{f_a} \frac{p_R - p_L}{\rho_{1/2}}\right)}{\partial \boldsymbol{Q}_L} =$$

$$\frac{K_p}{a^2_{1/2} f_a^2 \rho^2_{1/2}} \Bigg[\frac{\partial(\max(0, 1-\sigma \bar{M}^2))}{\partial \boldsymbol{Q_L}} (p_R - p_L)(f_a \rho_{1/2}) + \frac{\partial (p_R - p_L)}{\partial \boldsymbol{Q}_L} \max(0, 1-\sigma \bar{M}^2)(f_a \rho_{1/2}) - \frac{\partial f_a}{\partial \boldsymbol{Q}_L} \rho_{1/2} \left(\max(0, 1-\sigma \bar{M}^2)(p_R - p_L)\right) - \frac{\partial \rho_{1/2}}{\partial \boldsymbol{Q}_L} f_a \left(\max(0, 1-\sigma \bar{M}^2)(p_R - p_L)\right) \Bigg] \tag{A.25}$$

Now we need to compute each derivative term that appears in (A.25):

$$\frac{\partial(p_R - p_L)}{\partial \boldsymbol{Q}_L} = -\frac{\partial p_L}{\partial \boldsymbol{Q}_L} \tag{A.26}$$

$$\frac{\partial \rho_{1/2}}{\partial \boldsymbol{Q}_L} = \begin{pmatrix} 0.5 \\ \mathbf{0} \\ 0 \\ 0 \\ 0 \\ ... \\ 0 \end{pmatrix} \tag{A.27}$$

$$\frac{\partial(\max(0, 1 - \sigma \bar{M}^2))}{\partial \boldsymbol{Q_L}} = \begin{cases} -\sigma M_L \frac{\partial M_L}{\partial \boldsymbol{Q_L}}, & \text{if } 1 - \sigma \bar{M}^2 > 0 \\ 0, & \text{otherwise} \end{cases} \tag{A.28}$$

$$\frac{\partial f_a}{\partial \boldsymbol{Q}_L} = f(\bar{M}, M_{co}) \left(M_L \frac{\partial M_L}{\partial \boldsymbol{Q}_L} \right) \tag{A.29}$$

where

$$f(\bar{M}, M_{co}) = \begin{cases} 0, & \text{if } \bar{M}^2 > 1 \text{ or } \bar{M}^2 < M_{co}^2 \\ \frac{1-\bar{M}}{\bar{M}}, & \text{otherwise} \end{cases}$$

Analogous computations hold for the right state:

$$\frac{\partial M_P(M_L, M_R)}{\partial \boldsymbol{Q_R}} = \frac{K_p}{a_{1/2}^2} \frac{\partial \left(\frac{\max(0, 1-\sigma \bar{M}^2)}{f_a} \frac{p_R - p_L}{\rho_{1/2}} \right)}{\partial \boldsymbol{Q}_R} =$$

$$\frac{K_p}{a_{1/2}^2 f_a^2 \rho_{1/2}^2} \left[\frac{\partial(\max(0, 1 - \sigma \bar{M}^2))}{\partial \boldsymbol{Q_R}} (p_R - p_L)(f_a \rho_{1/2}) + \frac{\partial(p_R - p_L)}{\partial \boldsymbol{Q}_R} \max(0, 1 - \sigma \bar{M}^2)(f_a \rho_{1/2}) \right.$$
$$\left. - \frac{\partial f_a}{\partial \boldsymbol{Q}_R} \rho_{1/2} \left(\max(0, 1 - \sigma \bar{M}^2)(p_R - p_L) \right) - \frac{\partial \rho_{1/2}}{\partial \boldsymbol{Q}_R} f_a \left(\max(0, 1 - \sigma \bar{M}^2)(p_R - p_L) \right) \right] \tag{A.30}$$

with

$$\frac{\partial(p_R - p_L)}{\partial \boldsymbol{Q}_R} = \frac{\partial p_R}{\partial \boldsymbol{Q}_R} \tag{A.31}$$

$$\frac{\partial \rho_{1/2}}{\partial \boldsymbol{Q}_R} = \begin{pmatrix} 0.5 \\ \mathbf{0} \\ 0 \\ 0 \\ 0 \\ ... \\ 0 \end{pmatrix} \tag{A.32}$$

$$\frac{\partial(\max(0, 1 - \sigma \bar{M}^2))}{\partial \boldsymbol{Q_R}} = \begin{cases} -\sigma M_R \frac{\partial M_R}{\partial \boldsymbol{Q_R}}, & \text{if } 1 - \sigma \bar{M}^2 > 0 \\ 0, & \text{otherwise} \end{cases} \tag{A.33}$$

$$\frac{\partial f_a}{\partial \boldsymbol{Q}_R} = f(\bar{M}, M_{co}) \left(M_R \frac{\partial M_R}{\partial \boldsymbol{Q}_R} \right) \tag{A.34}$$

Let us focus now on the pressure related terms:

$$\frac{\partial p_{1/2}}{\partial \boldsymbol{Q}_L} = \mathcal{P}^+_{(5)} \frac{\partial p_L}{\partial \boldsymbol{Q}_L} + p_L \frac{\partial \mathcal{P}^+_{(5)}}{\partial \boldsymbol{Q}_L} - \frac{\partial P_P}{\partial \boldsymbol{Q}_L} \qquad \frac{\partial p_{1/2}}{\partial \boldsymbol{Q}_R} = \mathcal{P}^-_{(5)} \frac{\partial p_R}{\partial \boldsymbol{Q}_R} + p_R \frac{\partial \mathcal{P}^-_{(5)}}{\partial \boldsymbol{Q}_R} - \frac{\partial P_P}{\partial \boldsymbol{Q}_R} \tag{A.35}$$

$$\frac{\partial \mathcal{P}^{\pm}_{(5)}(M)}{\partial \boldsymbol{Q}} = \begin{cases} 0, & \text{if } |M| \geq 1 \\ \pm\frac{1}{4}\left(M \pm 1\right)\left(\pm 3 - 3M + 4\alpha\left(5M^2 - 1\right)\left(M \mp 1\right)\right)\frac{\partial M}{\partial \boldsymbol{Q}} \pm M(M^2-1)^2 \frac{\partial \alpha}{\partial \boldsymbol{Q}}, & \text{otherwise} \end{cases} \tag{A.36}$$

with

$$\frac{\partial \alpha}{\partial \boldsymbol{Q}} = \frac{15}{8} \frac{\partial f_a}{\partial \boldsymbol{Q}}$$

Therefore we get:

$$\frac{\partial P_P(M_L, M_R)}{\partial \boldsymbol{Q}_L} = K_u a_{1/2} \frac{\partial \left(\mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) \right)}{\partial \boldsymbol{Q}_L} =$$

$$\begin{aligned} K_u a_{1/2} \Bigg[& \frac{\partial \mathcal{P}^+_{(5)}}{\partial \boldsymbol{Q}_L} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \frac{\partial \mathcal{P}^-_{(5)}}{\partial \boldsymbol{Q}_L} \mathcal{P}^+_{(5)} (\rho_R + \rho_L) f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \\ & \frac{\partial (\rho_R + \rho_L)}{\partial \boldsymbol{Q}_L} \mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \frac{\partial f_a}{\partial \boldsymbol{Q}_L} \mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \\ & \frac{\partial (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n})}{\partial \boldsymbol{Q}_L} \mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) f_a \Bigg] \end{aligned} \tag{A.37}$$

and:

$$\frac{\partial P_P(M_L, M_R)}{\partial \boldsymbol{Q}_R} = K_u a_{1/2} \frac{\partial \left(\mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) \right)}{\partial \boldsymbol{Q}_R} =$$

$$\begin{aligned} K_u a_{1/2} \Bigg[& \frac{\partial \mathcal{P}^+_{(5)}}{\partial \boldsymbol{Q}_R} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \frac{\partial \mathcal{P}^-_{(5)}}{\partial \boldsymbol{Q}_R} \mathcal{P}^+_{(5)} (\rho_R + \rho_L) f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \\ & \frac{\partial (\rho_R + \rho_L)}{\partial \boldsymbol{Q}_R} \mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} f_a (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \frac{\partial f_a}{\partial \boldsymbol{Q}_R} \mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n}) + \\ & \frac{\partial (\boldsymbol{u}_R \cdot \boldsymbol{n} - \boldsymbol{u_L} \cdot \boldsymbol{n})}{\partial \boldsymbol{Q}_R} \mathcal{P}^+_{(5)} \mathcal{P}^-_{(5)} (\rho_R + \rho_L) f_a \Bigg] \end{aligned} \tag{A.38}$$

with

$$\frac{\partial(\rho_R+\rho_L)}{\partial \boldsymbol{Q}_L}=\frac{\partial(\rho_R+\rho_L)}{\partial \boldsymbol{Q}_R}=\begin{pmatrix}1\\ \mathbf{0}\\ 0\\ 0\\ 0\\ ...\\ 0\end{pmatrix} \tag{A.39}$$

$$\frac{\partial(\boldsymbol{u}_R\cdot\boldsymbol{n}-\boldsymbol{u}_L\cdot\boldsymbol{n})}{\partial \boldsymbol{Q}_L}=\begin{pmatrix}-\frac{\boldsymbol{u}_L\cdot\boldsymbol{n}}{\rho_L}\\ -\frac{\boldsymbol{n}}{\rho_L}\\ 0\\ 0\\ 0\\ ...\\ 0\end{pmatrix} \tag{A.40}$$

$$\frac{\partial(\boldsymbol{u}_R\cdot\boldsymbol{n}-\boldsymbol{u}_L\cdot\boldsymbol{n})}{\partial \boldsymbol{Q}_R}=\begin{pmatrix}\frac{\boldsymbol{u}_R\cdot\boldsymbol{n}}{\rho_R}\\ \frac{\boldsymbol{n}}{\rho_R}\\ 0\\ 0\\ 0\\ ...\\ 0\end{pmatrix} \tag{A.41}$$

Finally the derivatives of the Mach number with respect to the conserved variables are:

$$\frac{\partial M}{\partial \boldsymbol{Q}}=\begin{pmatrix}-\frac{M}{\rho}\\ \frac{\boldsymbol{n}}{\rho a_{1/2}}\\ 0\\ 0\\ 0\\ ...\\ 0\end{pmatrix} \tag{A.42}$$

A.5 Numerical Viscous Jacobians

For implicit time-stepping a Jacobian of viscous fluxes is required. Analytically, the viscous Jacobian depends on second order derivative information within the flow domain, which is difficult and expensive to acquire in general as it requires the use of non-local, "neighbour of neighbour" nodes. This can be avoided by applying the **Thin Shear Layer** (**TSL**) approximation [27], where dual-grid face tangential gradient information are neglected. Under TSL approximation, gradients of a generic scalar quantity ξ are the finite difference gradients between nodes i and j, projected into the cartesian directions,

$$\frac{\partial \xi}{\partial \cdot} = \frac{\xi_j - \xi_i}{d_{ij}} n_\cdot, \qquad \cdot = x, y, z$$

where d_{ij} is the distance between node i and node j.
Under the TSL approximation, only local quantities on opposing sides of the dual-grid interface are required, which makes the approximation well-suited to edge-based, unstructured solvers. Moreover in SU2 an other approximation is introduced for what concerns the Jacobian contribution due to momentum and energy balance equations: the value of the gradient at node j is computed using the mean value at interface and then its opposite is taken for node i, i.e:

$$\left.\frac{\partial \xi}{\partial \cdot}\right|_j = -\left.\frac{\partial \xi}{\partial \cdot}\right|_i = \frac{\bar{\xi}}{d_{ij}} n_\cdot, \qquad \cdot = x, y, z$$

where $\bar{\xi}$ is the mean value of a generic quantity ξ at face interface. Let us analyse first the contribution of the following part of the projected viscous flux:

$$\hat{\boldsymbol{G_{ij}}} = \begin{pmatrix} .. \\ \boldsymbol{\tau n} \\ (\boldsymbol{\tau u} - \boldsymbol{q}) \cdot \boldsymbol{n} \\ ... \end{pmatrix}, \tag{A.43}$$

If the transport coefficients are held fixed under differentiation and (A.43) is differentiated with respect to the primitive variables

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{u} \\ T \end{pmatrix}$$

the TSL Jacobian takes a simpler form,

$$\frac{\partial \hat{\boldsymbol{G_{i,j}}}}{\partial \boldsymbol{P}_{ij}} = \frac{\partial \hat{\boldsymbol{G_{i,j}}}^{(1)}}{\partial \boldsymbol{P}_{ij}} + \frac{\partial \hat{\boldsymbol{G_{i,j}}}^{(2)}}{\partial \boldsymbol{P}_{ij}}$$

where

$$\frac{\partial \hat{\boldsymbol{G_{ij}}}^{(1)}}{\partial \boldsymbol{P}_j} = \begin{bmatrix} \mu_{ij}\theta_x/d_{ij} & \mu_{ij}\eta_z/d_{ij} & \mu_{ij}\eta_y/d_{ij} & 0 \\ \mu_{ij}\eta_z/d_{ij} & \mu_{ij}\theta_y/d_{ij} & \mu_{ij}\eta_z/d_{ij} & 0 \\ \mu_{ij}\eta_y/d_{ij} & \mu_{ij}\eta_x/d_{ij} & \mu_{ij}\theta_z/d_{ij} & 0 \\ \mu_{ij}\pi_x/d_{ij} & \mu_{ij}\pi_y/d_{ij} & \mu_{ij}\pi_z/d_{ij} & \kappa_{ij} * \theta/d_{ij} \end{bmatrix} = -\frac{\partial \hat{\boldsymbol{G_{ij}}}^{(1)}}{\partial \boldsymbol{P}_i}$$

Herein μ_{ij} and κ_{ij} are the laminar viscosity and the thermal conductivity at the interface respectively, while the other quantities are defined as follows:

$$\begin{aligned}
\theta &= n_x^2 + n_y^2 + n_z^2 \\
\theta_x &= \theta + \frac{n_x^2}{3} \\
\theta_y &= \theta + \frac{n_y^2}{3} \\
\theta_z &= \theta + \frac{n_z^2}{3} \\
\eta_x &= \frac{n_y n_z}{3} \\
\eta_y &= \frac{n_z n_z}{3} \\
\eta_z &= \frac{n_x n_y}{3} \\
\pi_x &= \bar{u_x}\theta_x + \bar{u_y}\eta_z + \bar{u_z}\eta_y \\
\pi_y &= \bar{u_x}\eta_z + \bar{u_y}\theta_y + \bar{u_z}\eta_x \\
\pi_z &= \bar{u_x}\eta_y + \bar{u_y}\eta_x + \bar{u_z}\theta_z
\end{aligned}$$

where $\bar{u_\cdot}$, with $\cdot = x, y, z$ is the value of velocity at face interface.

Eventually

$$\frac{\partial \hat{\boldsymbol{G_{ij}}}^{(2)}}{\partial \boldsymbol{P}_j} = \frac{\partial \hat{\boldsymbol{G_{ij}}}^{(2)}}{\partial \boldsymbol{P}_i} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.5(\boldsymbol{\tau}_{ij}n)_x & 0.5(\boldsymbol{\tau}_{ij}n)_y & 0.5(\boldsymbol{\tau}_{ij}n)_z & \sum_{k=1}^{N_S} 0.5\boldsymbol{J}_{k_{ij}} \cdot \boldsymbol{n} C_{p_k}(\bar{T}) \end{bmatrix}$$

where $\bar{T}$ is the interface temperature.
The whole computation is very long to report; anyway in order to show the general procedure let us show the computation of $\frac{\partial(\boldsymbol{\tau n})_x}{\partial u_x}|_j$; first of all we recall that

$$(\boldsymbol{\tau n})_x = \mu_{ij}\left(2\frac{\partial \bar{u_x}}{\partial x} - \frac{2}{3}\nabla \cdot \bar{\boldsymbol{u}}\right) n_x + \mu_{ij}\left(\frac{\partial \bar{u_x}}{\partial y} + \frac{\partial \bar{u_y}}{\partial x}\right) n_y$$

Therefore

$$\left.\frac{\partial(\boldsymbol{\tau n})_x}{\partial u_x}\right|_j = \mu_{ij}\left(2\frac{\partial}{\partial u_x}\left(\frac{\partial \bar{u_x}}{\partial x}\right) - \frac{2}{3}\frac{\partial}{\partial u_x}\left(\nabla \cdot \bar{\boldsymbol{u}}\right)\right) n_x + \mu_{ij}\left(\frac{\partial}{\partial u_x}\left(\frac{\partial \bar{u_x}}{\partial y}\right) + \frac{\partial}{\partial u_x}\left(\frac{\partial \bar{u_y}}{\partial x}\right)\right) n_y$$

Changing the derivatives we get:

$$\left.\frac{\partial(\boldsymbol{\tau n})_x}{\partial u_x}\right|_j = \mu_{ij}\left(2n_x - \frac{2}{3}n_x\right)\frac{n_x}{d_{ij}} + \mu_{ij}\frac{n_y^2}{d_{ij}} = \mu_{ij}\frac{\left(\frac{4}{3}n_x^2 - n_y^2\right)}{d_i j} = \mu_{ij}\frac{\theta_x}{d_{ij}}$$

Since we need the derivatives with respect to the conserved variables, the following transformation matrix is also applied:

$$\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{Q}} = \begin{bmatrix} -\frac{u_x}{\rho} & \frac{1}{u_x} & 0 & 0 & 0 & 0 & 0 & .... & 0 \\ -\frac{u_y}{\rho} & 0 & \frac{1}{u_y} & 0 & 0 & 0 & 0 & .... & 0 \\ -\frac{u_z}{\rho} & 0 & 0 & \frac{1}{u_z} & 0 & 0 & 0 & .... & 0 \\ \frac{\partial T}{\partial \rho} & \frac{\partial T}{\partial \rho u_x} & \frac{\partial T}{\partial \rho u_y} & \frac{\partial T}{\partial \rho u_x} & \frac{\partial T}{\partial \rho E} & \frac{\partial T}{\partial \rho_1} & \frac{\partial T}{\partial \rho_2} & .... & \frac{\partial T}{\partial \rho_{N_S}} \end{bmatrix}$$

More attention is needed instead for the contribution due to species diffusion fluxes; indeed since the fluxes $\boldsymbol{J}_i, i = 1...N_S$ are defined implicitly through (2.18), it is not feasible to derive an analytic expression even with the use of TSL. Another approach could be the computation through a numerical Jacobian but even this case is too expensive from a computational point of view because it requires to solve one more set of Stefan-Maxwell equations for each interface and, above all, it necessitates the computation of gradient of molar fractions after perturbation which involves the costly algorithms of Green-Gauss or least squares.
Therefore the alternative exploited in order to compute an approximate Jacobian is to apply the TSL to Ramshaw self-consistent relations (2.13) keeping fixed the mean effective diffusion coefficients and the molar mass computed with the value of molar fractions at interface.
After some attempts, it has been noted that only the contribution due to single species guarantees the convergence of implicit Euler method; therefore let us report first the derivative of a generic molar fraction X_i with respect to a partial density ρ_k

$$\begin{aligned}\frac{\partial X_i}{\partial \rho_k} =& \frac{\partial \left(\sigma \frac{M}{M_i} Y_i\right)}{\partial \rho_k} = \frac{M}{M_i}\left(\frac{\partial \sigma}{\partial \rho_k} Y_i + \frac{\partial Y_i}{\partial \rho_k}\sigma\right) = \frac{M}{M_i}\left(\frac{Y_i}{\rho} + \delta_{ik}\frac{\sigma}{\rho}\right) \\ =& \frac{M}{M_i}\left(\frac{Y_i}{\rho}\frac{\sigma}{\sigma} + \delta_{ik}\frac{\sigma}{\rho}\right) = \frac{X_i}{\sigma\rho} + \frac{M}{M_i}\frac{\delta_{ik}}{\rho}\end{aligned}$$

If we denote by R and L the right and left state respectively we get:

$$\begin{aligned}\left.\frac{\partial \bar{\boldsymbol{J}}_i \cdot \boldsymbol{n}}{\partial \rho_k}\right|_R =& -\bar{\rho}\frac{M_i}{\bar{M}}\frac{\bar{D_{i,m}}}{d_{ij}}\left(\frac{X_{i,R}}{\sigma_R \rho_R} + \frac{M_R}{M_i}\frac{\sigma_R}{\rho_R}\delta_{ik}\right) + \bar{Y_i}\frac{\bar{\rho}}{\bar{M} d_{ij}}\sum_{j=1}^{N_S}\left(M_j \bar{D_{j,m}}\frac{X_{j,R}}{\sigma_R \rho_R}\right) + \bar{Y_i}\frac{\bar{\rho}}{\bar{M} d_{ij}}\frac{M_R \sigma_R}{\rho_R} \\ &+ \frac{1}{2}\frac{\delta_{ik}}{\rho_R}\sum_{j=1}^{N_S}\bar{\rho}\frac{M_j}{\bar{M}}\bar{D_{j,m}}\nabla \bar{X}_j, \qquad k = 1, ..., N_S \\ \left.\frac{\partial \bar{\boldsymbol{J}}_i \cdot \boldsymbol{n}}{\partial \rho_k}\right|_L =& \bar{\rho}\frac{M_i}{\bar{M}}\frac{\bar{D_{i,m}}}{d_{ij}}\left(\frac{X_{i,L}}{\sigma_L \rho_L} + \frac{M_L}{M_i}\frac{\sigma_L}{\rho_L}\delta_{ik}\right) - \bar{Y_i}\frac{\bar{\rho}}{\bar{M} d_{ij}}\sum_{j=1}^{N_S}\left(M_j \bar{D_{j,m}}\frac{X_{j,L}}{\sigma_L \rho_L}\right) - \bar{Y_i}\frac{\bar{\rho}}{\bar{M} d_{ij}}\frac{M_L \sigma_L}{\rho_L} \\ &+ \frac{1}{2}\frac{\delta_{ik}}{\rho_L}\sum_{j=1}^{N_S}\bar{\rho}\frac{M_j}{\bar{M}}\bar{D_{j,m}}\nabla \bar{X}_j, \qquad k = 1, ..., N_S\end{aligned}$$

where the symbol $-$ denotes a value taken at interface between state L and R.
In the previous derivative we neglect the contribution due to total mixture mass and moreover in computing the Jacobian we take at interface the arithmetic average of mass fractions.

It is worth to notice that for the diffusion fluxes we do not use the classical SU2 approximation of taking the mean value of interface for right state and then take the opposite for the left one, but we distinguish between the value at node L and the value at node R.

A.6 Source Chemistry Jacobian

Let us recall here the definition of the source chemistry term (2.35):

$$\dot{\omega}_i = M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right\}, \quad i = 1...N_S \tag{A.44}$$

We are interested in computing the Jacobian matrix related to these source terms in case the implicit time stepping strategy is employed; let us start from the derivative with respect to the mixture density ρ:

$$\begin{aligned}
\frac{\partial \dot{\omega}_i}{\partial \rho} &= \frac{\partial}{\partial \rho} \left(M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right\} \right) \\
&= M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ \frac{\partial k_{f,r}}{\partial \rho} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} - \frac{\partial k_{b,r}}{\partial \rho} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right\}
\end{aligned} \tag{A.45}$$

At this point we need to compute $\frac{\partial k_{f,r}}{\partial \rho}$ and $\frac{\partial k_{b,r}}{\partial \rho}$ and for this purpose we apply the chain rule, namely:

$$\frac{\partial k_{f,r}}{\partial \rho} = \frac{\partial k_{f,r}}{\partial T}\frac{\partial T}{\partial \rho} = \frac{k_{f,r}}{T}\left(\beta_r + \frac{T_r}{T}\right)\frac{\partial T}{\partial \rho} \tag{A.46}$$

$$\begin{aligned}
\frac{\partial k_{b,r}}{\partial \rho} &= \frac{\partial k_{b,r}}{\partial T}\frac{\partial T}{\partial \rho} = \frac{\partial \frac{k_{f,r}}{K_{eq}}}{\partial T}\frac{\partial T}{\partial \rho} = \frac{\left(\frac{\partial k_{f,r}}{\partial T} K_{eq} - \frac{\partial K_{eq}}{\partial T} k_{f,r}\right)}{K_{eq}^2}\frac{\partial T}{\partial \rho} \\
&= \frac{\frac{k_{f,r}}{T}\left(\beta_r + \frac{T_r}{T}\right) K_{eq} - k_{f,r}\frac{\partial K_{eq}}{\partial T}}{K_{eq}^2}\frac{\partial T}{\partial \rho} \\
&= \frac{k_{b,r}\left(\left(\frac{\beta_r}{T} + \frac{T_r}{T^2}\right) K_{eq} - \frac{\partial K_{eq}}{\partial T}\right)}{K_{eq}}\frac{\partial T}{\partial \rho}
\end{aligned} \tag{A.47}$$

Let us denote by $\gamma_r = \gamma_r(T) = \frac{\beta_r}{T} + \frac{T_r}{T^2}, \quad r = 1, ...N_R$; if we substitute (A.46) and (A.47) into (A.45) we obtain:

$$\frac{\partial \dot{\omega}_i}{\partial \rho} = \frac{\partial T}{\partial \rho} M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \gamma_r(T) - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \left(\gamma_r(T) - \frac{\frac{\partial K_{eq}}{\partial T}}{K_{eq}}\right) \right\} \tag{A.48}$$

Analogous relations hold for the derivatives with respect to the momentum and the total energy:

$$\frac{\partial \dot{\omega}_i}{\partial (\rho \boldsymbol{u})} = \frac{\partial T}{\partial (\rho \boldsymbol{u})} M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \gamma_r(T) - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \left(\gamma_r(T) - \frac{\frac{\partial K_{eq}}{\partial T}}{K_{eq}}\right) \right\} \tag{A.49}$$

$$\frac{\partial \dot{\omega}_i}{\partial \rho E} = \frac{\partial T}{\partial (\rho E)} M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \gamma_r(T) - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \left(\gamma_r(T) - \frac{\frac{\partial K_{eq}}{\partial T}}{K_{eq}}\right) \right\} \tag{A.50}$$

More attention is needed to compute $\frac{\partial \dot{\omega}_i}{\partial \rho_k}$:

$$\begin{aligned}
\frac{\partial \dot{\omega}_i}{\partial \rho_k} &= \frac{\partial}{\partial \rho_k}\left(M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right\} \right) \\
&= M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \frac{\partial}{\partial \rho_k} \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right\} \\
&= M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ \frac{\partial k_{f,r}}{\partial \rho_k} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} + \frac{\partial}{\rho_k} \left(\prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \right) k_{f,r} - \right. \\
&\qquad \left. \frac{\partial k_{b,r}}{\partial \rho_k} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} - \frac{\partial}{\partial \rho_k} \left(\prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right) k_{b,r} \right\} \\
&= M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \gamma_r(T) \frac{\partial T}{\partial \rho_k} + \frac{\partial}{\partial \rho_k} \left(\prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \right) k_{f,r} - \right. \\
&\qquad \left. k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \left(\gamma_r(T) - \frac{\frac{\partial K_{eq}}{\partial T}}{K_{eq}} \right) \frac{\partial T}{\partial \rho_k} - \frac{\partial}{\partial \rho_k} \left(\prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right) k_{b,r} \right\}
\end{aligned} \tag{A.51}$$

Let us analyse the term $\frac{\partial}{\partial \rho_k} \left(\prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \right)$:

$$\begin{aligned}
\frac{\partial}{\partial \rho_k} \left(\prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \right) &= \sum_{j=1}^{N_S} \left(\frac{1}{M_j} \phi'_{j,r} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}-1} \delta_{jk} \prod_{\substack{l=1 \\ l \neq j}}^{N_S} \left[\frac{\rho_l}{M_l}\right]^{\phi'_{l,r}} \right) \\
&= \sum_{j=1}^{N_S} \left(\frac{\phi'_{j,r}}{\rho_j} \delta_{jk} \prod_{l=1}^{N_S} \left[\frac{\rho_l}{M_l}\right]^{\phi'_{l,r}} \right) = \frac{\phi'_{k,r}}{\rho_k} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}}
\end{aligned} \tag{A.52}$$

Obviously the same holds with exponent $\phi''_{j,r}$; if we substitute (A.52) into (A.51) we eventually find:

$$\begin{aligned}
\frac{\partial \dot{\omega}_i}{\partial \rho_k} &= \frac{\partial T}{\partial \rho_k} M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} \gamma_r(T) - k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \left(\gamma_r(T) - \frac{\frac{\partial K_{eq}}{\partial T}}{K_{eq}} \right) \right\} \\
&+ M_i \sum_{r=1}^{N_R} \left(\nu''_{i,r} - \nu'_{i,r}\right) \left\{ \frac{\phi'_{k,r}}{\rho_k} k_{f,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi'_{j,r}} - \frac{\phi''_{k,r}}{\rho_k} k_{b,r} \prod_{j=1}^{N_S} \left[\frac{\rho_j}{M_j}\right]^{\phi''_{j,r}} \right\}
\end{aligned} \tag{A.53}$$

A.7 BiCGSTAB Algorithm

The biconjugate gradient stabilized gradient method, abbreviated as **BiCGSTAB**, is an iterative scheme for the numerical solution of linear system $\boldsymbol{Ax} = \boldsymbol{b}$ with $\boldsymbol{A}$ non-symmetric developed by H. A. van Der Vorst; it is a variant of the biconjugate gradient method (**BiCG**, see [30]) and therefore it belongs to the Krylov subspace methods.

The idea is to define residual vectors of the form :

$$\tilde{\boldsymbol{r}_i} = \Psi_i(\boldsymbol{A})\Phi_i(\boldsymbol{A})\boldsymbol{r}_0$$

instead of the definition:

$$\boldsymbol{r}_i = \Phi_i(\boldsymbol{A})\boldsymbol{r}_0$$

provided by standard BiCG with the hope that $\Psi_i(\boldsymbol{A})$ will enable faster and smoother convergence. Specifically $\Psi_i(\boldsymbol{A})$ is defined by simply recurrence:

$$\Psi_i(\boldsymbol{A}) = \prod_{j=1}^{i} (\boldsymbol{I} - \omega_j \boldsymbol{A})$$

with ω_j scalars to be determined.
Let us recall the recursive relations that characterize the BiCG algorithm:

$$\begin{aligned}\Phi_i(\boldsymbol{A}) &= \Phi_{i-1}(\boldsymbol{A}) - \alpha_i \boldsymbol{A}\Pi_{i-1}(\boldsymbol{A}) \\ \Pi_i(\boldsymbol{A}) &= \Phi_i(\boldsymbol{A}) + \beta_{i+1}\Pi_{i-1}(\boldsymbol{A})\end{aligned}$$

with α_i and β_i suitable coefficients we immediately obtain:

$$\Psi_i(\boldsymbol{A})\Phi_i(\boldsymbol{A})\boldsymbol{r}_0 = (\boldsymbol{I} - \omega_i\boldsymbol{A})\left(\Psi_{i-1}(\boldsymbol{A})\Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0 - \alpha_i \boldsymbol{A}\Psi_{i-1}(\boldsymbol{A})\Pi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0\right)$$

which entails for a recursive relation for the term $\Psi_i(\boldsymbol{A})\Pi_i(\boldsymbol{A})\boldsymbol{r}_0$; this can be also derived from BiCG:

$$\Psi_i(\boldsymbol{A})\Pi_i(\boldsymbol{A})\boldsymbol{r}_0 = \Psi_i(\boldsymbol{A})\Psi_i(\boldsymbol{A})\boldsymbol{r}_0 + \beta_{i+1}\left(\boldsymbol{I} - \omega_i\boldsymbol{A}\right)\Psi_{i-1}(\boldsymbol{A})\Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0$$

Moreover we define:

$$\tilde{\boldsymbol{p}_{i+1}} = \Psi_i(\boldsymbol{A})\Pi_i(\boldsymbol{A})\boldsymbol{r}_0$$

so that in vector form we get:

$$\begin{aligned}\tilde{\boldsymbol{p}_i} &= \tilde{\boldsymbol{r}_{i-1}} + \beta_i\left(\boldsymbol{I} - \omega_{i-1}\boldsymbol{A}\right)\tilde{\boldsymbol{p}_{i-1}} \\ \tilde{\boldsymbol{r}_i} &= \left(\boldsymbol{I} - \omega_i\boldsymbol{A}\right)\left(\tilde{\boldsymbol{r}_{i-1}} - \alpha_i\boldsymbol{A}\tilde{\boldsymbol{p}}\right)\end{aligned}$$

In order to derive a recursive relation for $\boldsymbol{x}_i$ we define:

$$\boldsymbol{s}_i = \tilde{\boldsymbol{r}_{i-1}} - \boldsymbol{A}\tilde{\boldsymbol{p}_i}$$

so that the recurrence relation for $\tilde{\boldsymbol{r}_i}$ becomes:

$$\tilde{\boldsymbol{r}_i} = \tilde{\boldsymbol{r}_{i-1}} - \alpha_i\boldsymbol{A}\tilde{\boldsymbol{p}_i} - \omega_i\boldsymbol{A}\boldsymbol{s}_i$$

and therefore since $\boldsymbol{r}_i = \boldsymbol{Ax}_i - \boldsymbol{b}$ we get:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \alpha_i\tilde{\boldsymbol{p}_i} + \omega_i\boldsymbol{s}_i$$

Now we need to determine the constants α_i and β_i and choose a suitable ω_i; in standard BiCG we define:

$$\begin{aligned}
\boldsymbol{r}_i &= \boldsymbol{A}\boldsymbol{x}_i - \boldsymbol{b} = \Phi_i(\boldsymbol{A})\boldsymbol{r}_0 \\
\hat{\boldsymbol{r}}_i &= \boldsymbol{A}^T\boldsymbol{x}_i - \boldsymbol{b} = \Phi_i(\boldsymbol{A}^T)\boldsymbol{r}_0 \\
\gamma_i &= (\hat{\boldsymbol{r}}_{i-1}, \boldsymbol{r}_{i-1}) = \left(\Phi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0\right)
\end{aligned}$$

where $(\cdot,\cdot)$ denotes the inner product and from these we impose:

$$\begin{aligned}
\alpha_i &= \frac{\gamma_i}{(\hat{\boldsymbol{p}}_i, \boldsymbol{A}\boldsymbol{p}_i)} \\
\beta_i &= \frac{\gamma_i}{\gamma_{i-1}}
\end{aligned}$$

with

$$\begin{aligned}
\boldsymbol{p}_i &= \boldsymbol{r}_{i-1} + \beta_i \boldsymbol{p}_{i-1} \\
\hat{\boldsymbol{p}}_i &= \hat{\boldsymbol{r}}_{i-1} + \beta_i \hat{\boldsymbol{p}}_{i-1}
\end{aligned}$$

Anyway, since BiCGSTAB does not explicitly keep track of $\hat{\boldsymbol{r}}_i$ or $\boldsymbol{r}_i$, we cannot directly use the previous formula for γ_i. However it can be related to the scalar

$$\tilde{\gamma}_i = \left(\Psi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0\right) = (\hat{\boldsymbol{r}}_0, \Psi_{i-1}(\boldsymbol{A})\Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0) = (\hat{\boldsymbol{r}}_0, \boldsymbol{r}_{i-1})$$

Due to biorthogonality $\boldsymbol{r}_{i-1} = \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0$ is orthogonal to $U_{i-2}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0$, where $U_{i-2}(\boldsymbol{A}^T)$ is any polynomial of degree $i-2$ ni $\boldsymbol{A}^T$. Hence only the highest order term of $\Phi_{i-1}(\boldsymbol{A}^T)$ and $\Psi_{i-1}(\boldsymbol{A}^T)$ are relevant in the scalar products $\left(\Phi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0\right)$ and $\left(\Psi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0\right)$. The leading coefficients of $\Phi_{i-1}(\boldsymbol{A}^T)$ and $\Psi_{i-1}(\boldsymbol{A}^T)$ are $(-1)^{i-1}\alpha_1\alpha_2...\alpha_{i-1}$ and $(-1)^{i-1}\omega_1\omega_2...\omega_{i-1}$ respectively. It follows that:

$$\gamma_i = \frac{\alpha_1\alpha_2...\alpha_{i-1}}{\omega_1\omega_2...\omega_{i-1}}\tilde{\gamma}_i$$

and

$$\beta_i = \frac{\gamma_i}{\gamma_{i-1}} = \frac{\tilde{\gamma}_i}{\tilde{\gamma}_{i-1}}\frac{\alpha_{i-1}}{\omega_{i-1}}$$

A similar approach can be used to determine α_i; since

$$\alpha_i = \frac{\gamma_i}{(\hat{\boldsymbol{p}}_i, \boldsymbol{A}\boldsymbol{p}_i)} = \frac{\Phi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0}{\Pi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \boldsymbol{A}\Pi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0}$$

only the highest order terms of $\Phi_{i-1}(\boldsymbol{A}^T)$ and $\Pi_{i-1}(\boldsymbol{A}^T)$ matter in inner products thanks to biorthogonality and biconjugacy and it happens that $\Phi_{i-1}(\boldsymbol{A}^T)$ and $\Psi_{i-1}(\boldsymbol{A}^T)$ have the same leading coefficient. Thus, replacing simultaneously with $\Psi_{i-1}(\boldsymbol{A}^T)$, we are led to:

$$\alpha_i = \frac{\Psi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \Phi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0}{\Psi_{i-1}(\boldsymbol{A}^T)\hat{\boldsymbol{r}}_0, \boldsymbol{A}\Pi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0} = \frac{\tilde{\gamma}_i}{\hat{\boldsymbol{r}}_0, \boldsymbol{A}\Psi_{i-1}(\boldsymbol{A})\Pi_{i-1}(\boldsymbol{A})\boldsymbol{r}_0} = \frac{\tilde{\gamma}_i}{(\hat{\boldsymbol{r}}_0, \boldsymbol{A}\tilde{\boldsymbol{p}}_i)}$$

Finally, BiCGSTAB selects ω_i to minimize $\tilde{\boldsymbol{r}}_i = (\boldsymbol{I} - \omega_i\boldsymbol{A})\,\boldsymbol{s}_i$ in 2-norm as function of ω_i; this happens when:

$$((\boldsymbol{I} - \omega_i\boldsymbol{A})\,\boldsymbol{s}_i, \boldsymbol{A}\boldsymbol{s}_i) = 0$$

giving the value:

$$\omega_i = \frac{(\boldsymbol{A}\boldsymbol{s}_i, \boldsymbol{s}_i)}{(\boldsymbol{A}\boldsymbol{s}_i, \boldsymbol{A}\boldsymbol{s}_i)}$$

A.8 Spline interpolation

Spline interpolation is a form of interpolation where the interpolant is a piecewise polynomial called **spline**.
One of the most famous example is represented by the **cubic spline**, whose formal definition is the following:

Given a set of $n+1$ data (x_i, y_i) where $x_i \neq x_j, \quad i,j = 0,...,n$ and $a = x_0 < x_1 < ... < x_n = b$, the cubic spline $S(x)$ is a function satisfying:

1. $S(x) \in C^2([a,b]$
2. On each subinterval $[x_{i-1}, x_i], \quad i = 1,...,n$ $S(x)$ is a third order polynomial
3. $S(x_i) = y_i, \qquad i = 0,...,n$

Therefore we get:

$$S(x) = \begin{cases} P_1(x), & x_0 \leq x \leq x_1 \\ ... & \\ P_i(x), & x_{i-1} < x \leq x_i \\ ... & \\ P_n(x), & x_{n-1} < x \leq x_n \end{cases}$$

where each $P_i = a_i + b_i x + c_i x^2 + d_i x^3, i = 1,...,n$ is a cubic function.

We need to determine $a_i, b_i, c_i, d_i i = 1,...,n$ imposing:

1. $P_i(x_{i-1}) = y_{i-1}$ and $P_i(x_i) = y_i, \qquad i = 1,...,n$
2. $P_i'(x_i) = P_{i+1}'(x_i), \qquad i = 1,...,n-1$
3. $P_i''(x_i) = P_{i+1}''(x_i), \qquad i = 1,...,n-1$

We can see that there $4n-2$ conditions, but we need to determine $4n$ coefficients, therefore we need to add two boundary conditions.

Three are the types of common boundary conditions:

1. First derivatives at the endpoints are known;
2. Second derivatives at the endpoints are known. In the special case they are both 0 this is known as **natural** or simple boundary condition;
3. When the exact function we want to interpolate is a periodic function with period $x_n - x_0$, $S(x)$ is a periodic function of period $x_n - x_0$ too.

In this work we employ null first derivatives at endpoints.

A little side calculation shows that there is only one way to arrange the polynomial P_i in order to highlight the contribution due to second order derivative:

$$P_i(x) = Ay_{i-1} + By_i + Cy_{i-1}'' + Dy_i'' \tag{A.54}$$

where

$$A = \frac{x_i - x}{x_{i-1} - x_i}, \qquad B = 1 - A = \frac{x - x_{i-1}}{x_i - x_{i-1}} \qquad C = \frac{1}{6}(A^3 - A)(x_i - x_{i-1}) \qquad D = \frac{1}{6}(B^3 - B)(x_i - x_{i-1})$$

If we take the first derivative of (A.54) we obtain:

$$P'(x) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} - \frac{3A^2 - 1}{6}(x_i - x_{i-1})y''_{i-1} + \frac{3B^2 - 1}{6}(x_i - x_{i-1})y''_i \tag{A.55}$$

At this point if, in the equation (A.55), we impose that the first derivative evaluated at $x = x_i$ in the interval (x_{i-1}, x_i) is the same of the first derivative evaluated at at $x = x_i$ in the interval (x_i, x_{i+1}), we find the relations to compute the second order derivatives.

For further details, please refer to [36].

A.9 Secant Method

The secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function $f(x)$.
Starting with initial values x_0 and x_1, the recurrence relation is defined as:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2, ...$$

If the function f is twice continuously differentiable, the root in question is simple and the initial values x_0 and x_1 are sufficiently close to the root, the iterates x_n converge to a root of f with order of convergence $\phi = \frac{1+\sqrt{5}}{2}$.

It is also a **quasi-Newton** method because it can be derived from the Newton's formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 1, 2....$$

by replacing $f'(x_n)$ with a finite-difference approximation:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

If we compare Newton's method with the secant method, we see that Newton's method converges faster (order 2 against $\phi \approx 1.6$), but Newton's method requires the evaluation of both f and its derivative while the secant method requires only the evaluation of f.

Bibliography

[1] Numerical Modeling and Simulations of Combustion Processes in Hybrid Rocket Engines, Mazzetti, Alessandro

[2] SU2 Detailed documentation: `https://su2code.github.io/docs/home/`

[3] R. A. Svehla: Transport Coefficients for the NASA Lewis Chemical Equilibrium Program, NASA Technical Memorandum 4647, NASA-TM-4647, National Aeronautics and Space Administration Lewis Research Center, Cleveland, Ohio; USA; 1995; available online at `https://www.grc.nasa.gov/www/CEAWeb/TM-4647.pdf`

[4] Burcat, A., Ruscic, B., Third Millennium Ideal Gas and Condensed Phase Thermochemical Database for Combustion with Updates from Active Thermochemical Tables, Argonne Report ANL 05/20 and Technion Aerospace Report TAE N. 960, September 2005.

[5] Multi-Component Diffusion with Application To Computational Aerothermodynamics, Sutton, Kenneth and Gnoffo, Peter, 02/1998

[6] Venkateswaran, S., Merkle, C.L., Size Scale-Up in Hybrid Rocket Motors, AIAA Aerospace Sciences Meeting and Exhibit, 34th, Reno, NV; USA; 15-18 January, 1996, AIAA Paper 1996-0647.

[7] Jones,W.P., Lindstedt, R.P., Global Reaction Schemes for Hydrocarbon Combustion, Combustion and Flame, 73(3): 233 - 249, 1988.

[8] Arisawa, H., Brill, T.B., Flash Pyrolisis of Hydroxil-Terminated Polybutadiene (HTPB) I: Analysis and Implications of the Gaseous Products, Combustion and Flame 106: 131 - 143, 1996.

[9] Arisawa, H., Brill, T.B., Flash Pyrolisis of Hydroxil-Terminated Polybutadiene (HTPB) II: Implications of the Kinetics to Combustion of Organic Polymers, Combustion and Flame 106: 144 - 154, 1996.

[10] Kuo, K.K., Principles of Combustion, 2nd Ed., Chapter 3. John Wiley & Sons, New York, NY; 2005.

[11] Chung, T.H., Lee, L.L., Starling, K.E., Applications of Kinetic Gas Theories and Multiparameter Correlation for Prediction of Dilute Gas Viscosity and Thermal Conductivity, Industrial and Engineering Chemical Fundamentals, 23(1): 8 - 13, 1984.

[12] Fuller, E.N., Schettler, P.D., Giddings, J.C., A new method for prediction of binary gas-phase diffusion coefficients, Industrial and Engineering Chemistry, 58(5):19-27, 1966.

[13] Quarteroni, A., Numerical Models for Differential Problems,2014

[14] Praaven, C., Finite volume method on unstructured grid, 2013, `http://math.tifrbng.res.in/~praveen/notes/acfd2013/fvm.pdf`

[15] Veynante, D., Poinsot, T., Theoretical and Numerical Combustion, R.T. Edwards, P.O. Box 27388, Philadelphia, PA, 19118; USA; 2001.

[16] Turns, S.R., An introduction to combustion: Concepts and Applications, 2nd edition, McGraw-Hill, New York, 2000

[17] Liou, M.-S. and Steffen, C., A New Flux Splitting Scheme, Journal of Computational Physics, 107: 23-39, 1993.

[18] Liou, M.-S., A Sequel to AUSM: AUSM+, Journal of Computational Physics, 129: 364-382, 1996.

[19] Liou, M.-S., A Sequel to AUSM, Part II: AUSM+-up, Journal of Computational Physics, 214: 137- 170, 2006.

[20] Wilke, C.R., A Viscosity Equation for Gas Mixtures, The Journal of Chemical Physics, 18(4): 517 - 519, 1950.

[21] Green, D.W., Perry, R.H., Perry's Chemical Engineer Handbook, 8th Ed., McGraw-Hill, 2008.

[22] Stiel, L.I., Thodos, G., The Thermal Conductivity of Nonpolar Substances in the Dense Gaseous and Liquid Regions, American Institute of Chemical Engineers Journal, 10(1): 26 - 30, 1964.

[23] Mason, E.A., Saxena, S.C., Approximate Formula for the Thermal Conductivity of Gas Mixtures, The Physics of Fluids, 1(5): 361 - 369, 1958.

[24] Giovangigli, V., Convergent iterative methods for multicomponent diffusion, IMPACT Comput. Sci. Eng., 1990, Vol. 2, 73-97

[25] Giovangigli, V., Mass Conservation and Singular Multicomponent Diffusion Algorithms, IMPACT Comput. Sci. Eng., 1991, Vol. 3, 244-276

[26] Peerenboom K.S.C, Van Dijk J., Ten Thije Boonkkamp J.H.M, Liu L., Goedheer W.J., Van Der Mullen J.J.A.M., Mass conservative finite volume discretization of the continuity equations in multi-component mixtures, Journal of Computational Physics 230 (2011) 3525?3537

[27] Blazek, J., Computational Fluid Dynamics: Principles and Applications, Elsevier Science, 2001

[28] Valdettaro, L., Dispense del corso di Teoria, Modellistica e Simulazione della Turbolenza, Allievi di Ingegneria Matematica - Laurea Specialistica -Indirizzo Scienze Computazionali per l'Ingegneria A.A. 2007/08

[29] Wilcox, D.C., Turbulence Modeling for CFD, 3rd Ed. Chapter 5. DCW Industries, 5354 Palm Drive, La Canada, California, 91011; USA; 2006.

[30] Saad, Y., Iterative methods for sparse linear systems, Second Edition, Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2003

[31] Andersen J., Rasmussen L. C., Giselsson T., Glarborg P., Global Combustion Mechanisms for Use in CFD Modeling under Oxy-Fuel Conditions, Energy & Fuels, 2009, 23, 1379-1389

[32] Venkatakrishnan V., Convergence to steady state solutions of the Euler equations on unstructured grids with limiters, Journal of Computational Physics, 118:120-130, 1995.

[33] Barth T. J. and Jespersen D.C., The design and application of upwind schemes on unstructured meshes, In AIAA Paper 89-0366, Reno(NV), Jan 1989, 37th AIAA Aerospace Science Meeting and Exhibit.

[34] Ames Research Staff, Equations, tables and charts for compressible flows, Ames Aeronautical Laboratory, Moffett Field, California, Report 1135, 1953

[35] Roop N. Gupta, Jerrold M. Yos, Richard Thompson, Kam-Pui Lee, A review of reaction rates and thermodynamic and transport properties for an 11-species air model for chemical and thermal nonequilibrium calculations to 30000 K, 1990

[36] Press, William H.; Teukolsky, Saul A.; Vetterling, William T.; Flannery, Brian P., Numerical Recipes. The Art of Scientific Computing, 3rd Edition, 2007

[37] Wesseling P., Principles of Computational Fluid Dynamics, Springer, 2001, 518-524

[38] Barbante, P. F., Accurate and Efficient Modelling of High Temperature Nonequilibrium Air Flows, PhD Thesis, Université libre de Bruxelles, Von Karman Institute

[39] Bottin, B., Vander Abele D., Carbonaro M., Degeez G., Sarma G. S. R., hermodynaimic and Tranposrt Properties for Inductive Plasma Modeling, J.of Thermophyisics and Heat Transfer, 13,3(1999),343-350