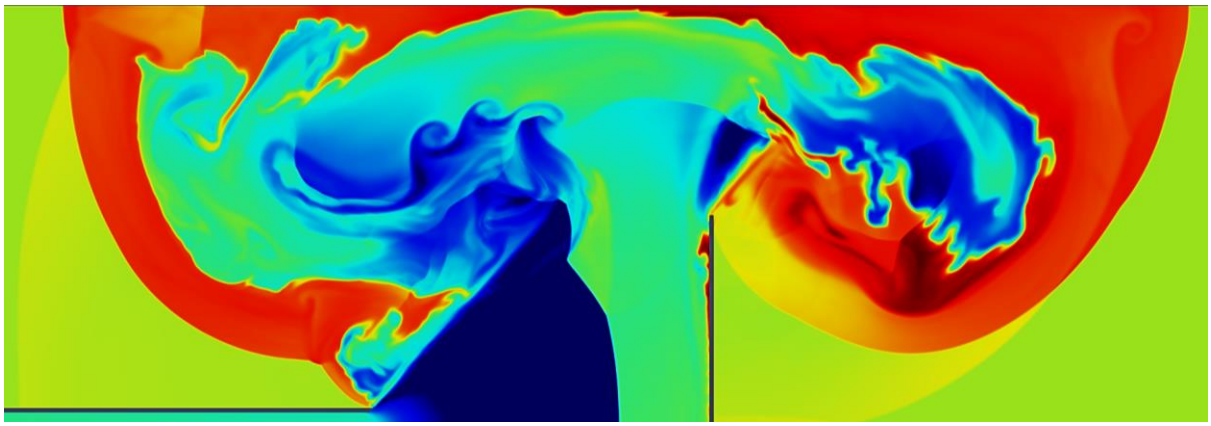
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2020

Energy and Environmental Technology

Release of high-pressure hydrogen into the air



Keivan Afshar Ghasemi

Faculty of Technology, Natural sciences and Maritime Sciences

Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2020

Title: Release of high-pressure hydrogen into the air

Number of pages: 118

Keywords: High-pressure Hydrogen, Compressed Gaseous Hydrogen (CGH2), Spontaneous-ignition, Leakage, Numerical Investigation, CFD Simulation, USN-FLIC Code, OpenFOAM, Riemann Problem, Shock-tube Problem

Student:	Keivan Afshar Ghasemi
Supervisors:	Prof. Dag Bjerketvedt, Prof. Em. Are Mjaavatten, and Prof. Knut Vågsæther
External partner:	FME - MoZEES (www.mozees.no)
Availability:	Open

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Release of high-pressure hydrogen into the air

Student: Keivan Afshar Ghasemi

USN supervisors: Prof. Dag Bjerketvedt, Prof. Em. Are Mjaavatten and Prof. Knut Vågsæther

External partner and supervisor: FME-MoZEES (www.mozees.no)

Task background:

A high-pressure release of hydrogen into the air will generate a complex flow with shock waves and expansion waves. Such releases have the potential of spontaneous ignition. However, the mechanisms causing the ignition are not well understood.
This thesis aims to study how shock waves and mixing can generate conditions favorable for ignition when hydrogen is accidentally released into the air. This work is associated with hydrogen safety issues in MoZEES RA3 (www.mozees.no) and other USN projects.

Task description:

- Literature review on the high-pressure release of hydrogen into the air
- Use the USN-FLIC and Open-FOAM codes to simulate the flowfield for high-pressure release of hydrogen into the air. Evaluate the conditions for ignition based on predicted the state of the flow field.
- Optional: Write a draft paper for the 2021 ICHS (International Conference on Hydrogen Safety) on the subject.

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes the necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc.).

Signatures:

Supervisor: 2020-10-12

Student: 2020-10-12

University of South-Eastern Norway

www.usn.no

Summary:

Hydrogen as an energy carrier is one of the available alternatives to fossil fuels for decarbonizing the global energy system. Regarding to the very low volumetric energy content of gaseous hydrogen, for practical and economical storage, it has to be either pressurized at ambient temperature (Compressed Gaseous Hydrogen – CGH2) or liquefied at cryogenic temperatures (Liquid Hydrogen – LH2). On the subject of CGH2, using high-pressure hydrogen reservoirs from $250\ MPa$ up to $70\ MPa$ is conventional; and among these, utilizing $70 - MPa$ hydrogen reservoirs as fuel cells in transportation or as storage tanks, for instance, in local hydrogen refueling stations is the common approach.

Having a leakage in such a high-pressure reservoir will form shock waves in front of the released hydrogen causing a temperature rise which may be intensified by existing obstacles in that environment; the presence of these obstacles and confinement may also enhance the hydrogen-air mixing. Because of the wide flammability range of hydrogen, this might lead to its spontaneous ignition if these high-temperature, well-mixed regions could last as long as the hydrogen-air induction time. In this thesis, it is tried to numerically investigate the possibility of this kind of scenario.

In order to simulate a $70 - MPa$ hydrogen release into the air (treated as a dual pseudo species), in the initial attempts, the USN-FLIC code was tried, but because the results were not convincing, it was decided to use the OpenFOAM software as an alternative. Considering the restrictions of the solvers of OpenFOAM (v.7), combinations of solvers along with new thermophysical models are used to be able to overcome the so-called Riemann problem of shock waves in a non-ideal, multi-component, non-reacting mixture. Furthermore, to validate this method, the shock-tube problem is solved and the results are compared with available data of similar cases.

According to the results, potentially hazardous regions are formed in the domain that are mostly related to the interaction of the reflected, rarefaction, and normal shock waves inside the flow field. Although there are some inconsistencies between the results of the simulations in estimating the flow properties, generally, the risk of hydrogen auto-ignition in these regions is quite high. But for having a better understanding about the effects of distances between wall boundaries, it is shown that the simulation should be done in three-dimensions.

Preface

The current report presents the observation of the master's thesis with the title "release of high-pressure hydrogen into the air" carried out by using the USN-FLIC code and OpenFOAM software. For all the technical and emotional support, and guidance throughout my work on this thesis, particularly under these exceptional circumstances caused by COVID-19, I would like to sincerely thank my supervisors, especially Prof. Dag Bjerketvedt. In addition, I would like to appreciate all my family and friends who supported me either way during this period.

Porsgrunn, Norway, 2021-02-15

Keivan Afshar Ghasemi

Contents

1 Introduction 10

1.1 Chemical and Physical Aspects of Hydrogen Hazard 10
1.2 High-Pressure Hydrogen Release 11
1.2.1 Experimental Studies 12
1.2.2 Numerical Studies 13
1.2.3 Combined Studies 15
1.3 Defining the Present Project Scenario 17

2 CFD Background 18

2.1 Conservation Laws and Characteristic Curves 18
2.2 The Finite Volume Method (FVM) 20
2.3 Non-Uniqueness of Solutions and the Entropy Condition 20
2.4 Relevant Riemann Solvers to This Project 22
2.4.1 The TVD Schemes 23
2.4.2 The FLIC Scheme 23
2.4.3 The MUSCL Scheme 23

3 USN-FLIC Code Simulations 25

3.1 The Code Properties and Governing Equations 25
3.2 Simulation Setup 26
3.2.1 Geometry and Boundary Conditions 26
3.2.2 Mesh 27
3.2.3 Initial Conditions 29
3.3 Simulation Results and Discussion 29

4 OpenFOAM Simulations 34

4.1 Solver Selection 34
4.1.1 Large gradients and Mass Convection 34
4.1.2 Compressibility 35
4.1.3 Multi-Component Mixtures 36
4.1.4 Final Decision 36
4.2 Discovered Solvers and The Properties of the Chosen One 37
4.2.1 rhoReactingCentralFoam Governing Equations 37
4.3 Implementation of a New Thermophysical Model 39
4.4 Initial Simulations Setups 40
4.4.1 Geometry and Boundary Conditions 41
4.4.2 Mesh 42
4.4.3 Initial Conditions 43
4.4.4 System Sub-Dictionaries 43
4.4.5 fvSchemes 44
4.4.6 fvSolution 44
4.4.7 Constant Sub-Dictionaries 45
4.5 Initial Simulations Results and Discussion 48
4.6 Validation of the Method and the Shock-Tube Problem 53
4.6.1 Comparison the Results 54
4.7 Implementation of the New Thermodynamic and Transport Models 59
4.7.1 nasa9Poly Thermodynamic Model 59
4.7.2 nasaPoly Transport Model 59
4.8 Generating New Sets of Coefficients for the New Models 60
4.9 Shock-Tube Simulation with the New Thermophysical Model 63

4.10 Final Simulation Results and Discussion 65
5 Conclusion 69
6 Recommendations 71
7 References 72
8 Appendices 76

Nomenclature

Abbreviations

USN	University of South-Eastern Norway
CGH2	Compressed Gaseous Hydrogen
LH2	Liquid Hydrogen
MIE	Minimum Ignition Energy
CFD	Computational Fluid Dynamics
FLACS	Flame Acceleration Simulator
TVD	Total Variation Diminishing
MUSCL	Monotonic Upstream-centered Scheme for Conservation Laws
ALE	Arbitrary Lagrangian-Eulerian method
LES	Large-Eddy Simulation
ODE	Ordinary Differential Equation
PDF	Partial Differential Equation
FVM	Finite Volume Method
FDM	Finite Difference Method
FLIC	Flux-Limiter Centered scheme
OpenFOAM	Open-source Field Operation and Manipulation
OF	OpenFOAM
EOS	Equation of State
PBiCGStab	Preconditioned Bi-conjugate Gradient Stabilized solver
CJ	Chapman-Jouguet theory

Dimensionless quantities

Re	Reynolds number
Pr	Prandtl number
Da	Dahmkohler number
Pe	Peclet number

Symbols

u_i	Velocity
t	Time

$\vec{f}$	Flux
S	Source term
a	Shock speed
x	Space variable
Ω	Control volume
$\partial\Omega$	Control surface
TV	Total variation
τ	Induction time
$A_\alpha, B_\alpha, C_\alpha, D_\alpha$	Coefficients of the induction time equation
S_L	Laminar flame velocity
S_T	Turbulent flame velocity
T_0	Initial temperature
T	Temperature
p_0	Initial pressure
p	Pressure
γ	Isentropic expansion factor
c_p	Specific heat capacity at constant pressure
c_v	Specific heat capacity at constant volume
MW	Molecular weight
R	Universal gas constant
ρ	Density
$x_{H_2}, x_{O_2}, x_{N_2}$	Molar concentrations
ϕ	Flow property
Γ	Diffusion coefficient
L	Characteristic length
D	Mass diffusion coefficient
μ	Dynamic viscosity
k	Thermal conductivity
α	Thermal diffusivity
Z	Compressibility factor
n	Number of moles
I, V	Inviscid, Viscous

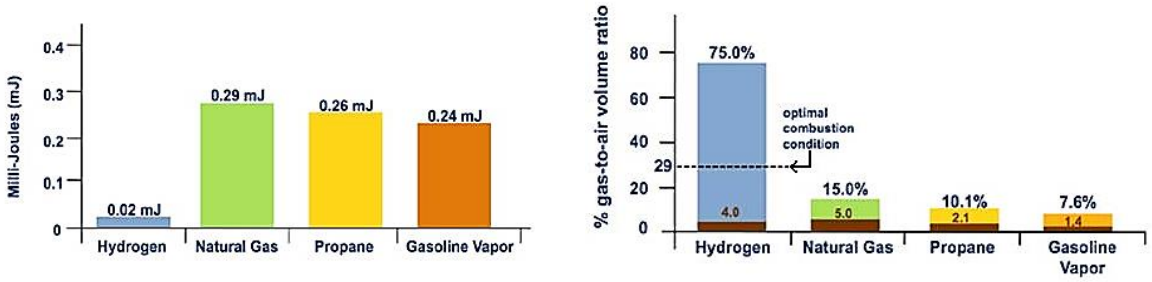
Y_i	Mass fraction of species i
E	Total Energy
R_i	Reaction rate of species i
Ψ	Compressibility
A_s, T_s	Sutherland coefficients
H	Enthalpy
S	Entropy
k	Turbulent kinetic energy
ω	Specific dissipation rate
ε	Turbulent dissipation
C_μ	Coefficient for calculating specific dissipation rate
I	Flow intensity
C_p^0	ideal-gas specific heat capacity at constant pressure
H^0	ideal-gas enthalpy
S^0	ideal-gas entropy

1 Introduction

These days that human societies struggling with the challenge of meeting the growing energy demand while addressing the global warming concerns, hydrogen is among the best solutions that we have for carrying and storing clean renewable energy to decarbonize the global energy system. However, because of the characteristics of this chemical substance, at least during the past one hundred years, there have always been reports of major hydrogen explosions due to leakage or even during an operation such as depressurizing vessels [1], [2]. In the following, the scientific principles behind this hazard as well as some studies that were conducted to improve our fundamental understanding of such phenomenon will be investigated. At last, also, the scenario of this project will be defined.

1.1 Chemical and Physical Aspects of Hydrogen Hazard

From the chemical aspect point of view, on the one hand, as depicted in *Figure 1.1*, the minimum ignition energy of gaseous hydrogen (at 29% hydrogen-air volumetric ratio) is much lower than the other common fuels like natural gas, on the other hand, its flammability range is much wider than the others; this combination makes hydrogen violently combustible.



(a) (b)

Figure 1.1: (a) Minimum ignition energy (MIE) (b) Volumetric flammability limit range of different fuels [3]

Although as it is discussed in *Appendix A*, the standard condition values indicated in the above diagrams are highly affected by the pressure, temperature, and also concentration of reactants (hydrogen and air), evaluations of the hydrogen explosion hazard have generally shown that the risk is quite high; as mentioned in the work of Bain et al., statistically, around 80% of industrial hydrogen leaks eventually ignite [4].

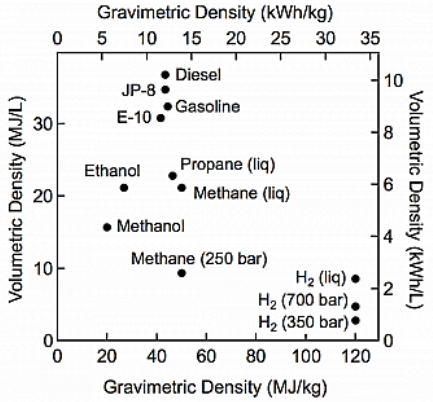


Figure 1.2: Comparison between the gravimetric energy density (energy content per mass) and the volumetric energy density (energy content per volume) of several fuels (based on lower heating values) [5]

From another point of view, as seen in *Figure 1.2*, because of the low volumetric energy content of the gaseous hydrogen in comparison to other traditional fossil fuels, during the past century along with technological advancements in hydrogen storing, it has always been tried to store hydrogen more efficiently. As a result of this effort, with the physical-based storage approach[1], the pressure of the hydrogen reservoirs has become higher and higher through this period which potentially makes them more and more dangerous [5]–[7].

In *Figure 1.3*, the pressure, temperature, and density ranges of hydrogen reservoirs under different storage conditions are demonstrated. Nowadays, the first two methods, i.e. liquefying at cryogenic temperatures (Liquid Hydrogen – LH2) and pressurizing at ambient temperature (Compressed Gaseous Hydrogen – CGH2), are the conventional ones for everyday usage.

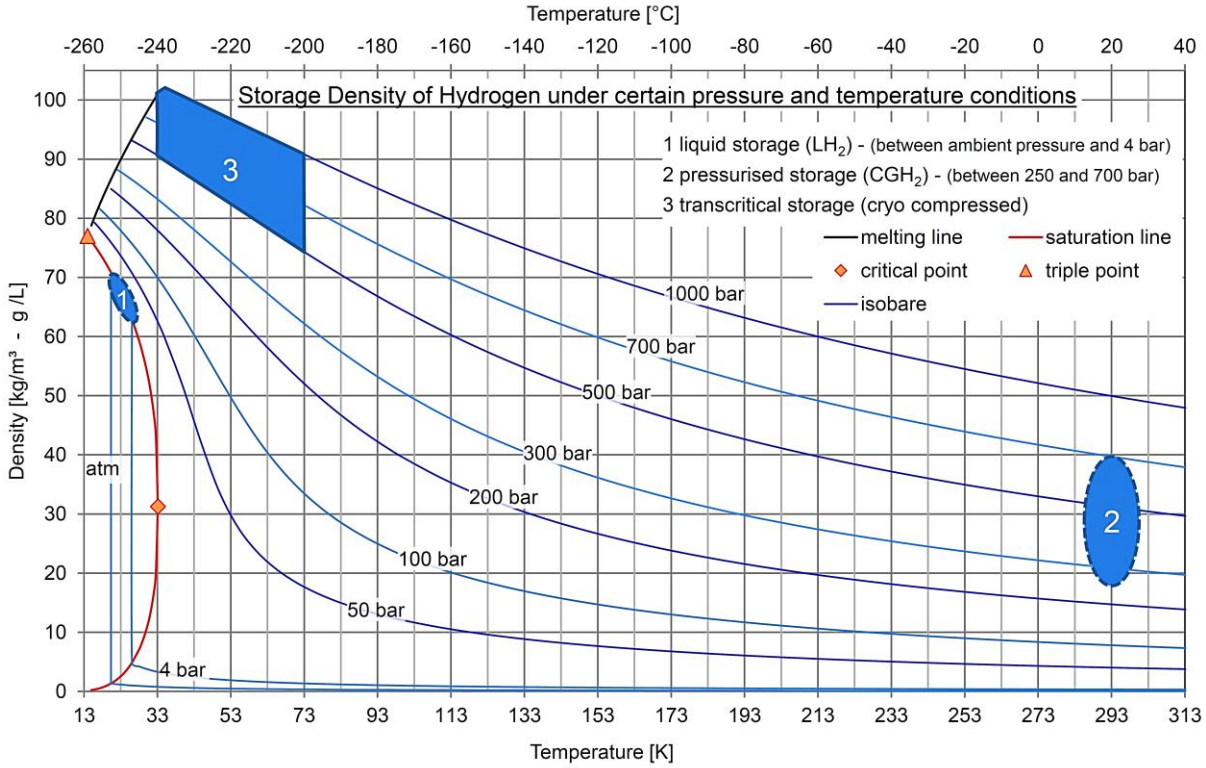


Figure 1.3: Storage density of hydrogen under certain pressure and temperature conditions [7]

1.2 High-Pressure Hydrogen Release

On the subject of compressed gaseous hydrogen which is the focus of this project, as it was mentioned, there have been many reports about the high-pressure hydrogen release throughout the history of hydrogen consumption that have continued even until recently. According to Astbury and Hawksworth, among the gaseous hydrogen releases that lead to an explosion, in

[1] In general, two main storage approaches have been pursued during this time, the physical-based storage and the material-based storage.CGH2, LH2, and cryo-compressed are the subcategories of the physical-based one. On the other side, adsorption-based storage (storing hydrogen on the surface of solids) and absorption-based storage (storing hydrogen within solids) are the subcategories of the material-based one [5]. For more details, you can look at *Figure B.1* in *Appendix B*.

only 13.6% of cases, the source of ignition was identified which is around 21% less than that of non-hydrogen releases [1], [2]. This statement clearly specifies why numerous studies have been conducted on the subject of finding hydrogen ignition sources. Apart from some exceptional studies which have tried to investigate and explain the desired hydrogen explosion analytically, these researches are categorized into 3 main categories which will be discussed from section *1.2.1* to *1.2.3*. Also based on the conventional classification done by Astbury and Hawksworth, the postulated mechanisms of hydrogen ignition can be divided into 5 main groups [2]:

- Reverse Joule-Thomson Effect
- Electrostatic Ignition
- Diffusion Ignition
- Sudden Adiabatic Compression
- Hot Surface Ignition

The subject that will be covered by this project, is basically related to diffusion ignition. This term was first suggested by Wolanski and Wojcicki in 1972 [8]. And it refers to a situation in which releasing high-pressure hydrogen into an oxidizer leads to forming shock waves in front of the hydrogen jet. These shock waves increase the temperature at the contact surface and because of the heat and mass diffusion to this hydrogen-oxidizer mixture, the temperature increment could finally ignite this combustible mixture without any external source [1], [8].

1.2.1 Experimental Studies

The first category, which specifically the oldest investigations were carried out based on, is the experimental method. In this category, initially, it is tried to investigate the incident and find the possible explosion scenarios, after that the researchers try to make laboratory models of those scenarios and examine each of them to find the most possible reason behind the explosion. Also, there are cases that only hypothetical setups are examined in the laboratory to find the real ignition scenario of those cases to improve our fundamental knowledge of hydrogen explosion. Among the latter kind of studies, the work of Dryer et al. (2007) is quite unique. Because it was one of the first researches that try to investigate high-pressure hydrogen releases into usual kinds of workplace environments and "*determine the envelope of design parameters*"[1] for those places [1], [9]. This research is particularly relevant to the present thesis since, in contrast to other reviewed studies, the effect of the downstream obstructions is included in it.

Despite all major incidents and their subsequent experimental investigations listed in the HSE spontaneous ignition of hydrogen literature review (2008) that apparently most of them happened due to an electrostatic ignition or a spark caused by a collision, Dryer et al. showed in their project that a high-pressure hydrogen release may lead to spontaneous ignition, under certain circumstances [1]. According to Dryer et al., the main parameters that provide a short mixing time scale in the contact surface which is a prerequisite of a diffusion ignition are the "*pressure boundary failure geometry, multi-dimensional shock-boundary, and shock-shock*

[1] The parameters that should be taken into account when an industrial area, in where high-pressure hydrogen vessels are going to be used, is being designed; such as the temperature and the pressure of that environment, or the arrangement of the devices which corresponds to the downstream obstructions that will be discussed.

interactions in addition to molecular diffusion". Also, it is worth mentioning that the main achievement of their work was to show the importance of the effect of geometry on the explosion enhancement, both the aperture geometry and the downstream obstructions. Based on that, the minimum required pressure of compressed gaseous hydrogen for having a spontaneous ignition is "*dictated by the reflected shock and shock-shock interactions*" which are affected by the geometry [9]. However, in general, the auto-ignition was only observed in cases with a release pressure of more than 20 bar [8].

1.2.2 Numerical Studies

The second category, which specifically the recent investigations mostly were carried out based on, is the numerical method. In this category, the scenarios of real or hypothetical cases are studied by numerical methods, or more accurately, it is tried to simulate the cases by computational fluid dynamics (CFD) methods. The present project approach is also based on this method. The main obstacle in the way of the numerical simulation of a high-pressure hydrogen release is the so-called Riemann problem which will be discussed in the next chapter. To overcome this problem, several numerical methods and CFD software have been proposed that will be mentioned in the following.

One of the most recent incidents, which was investigated by the Gexcon[1] company, is an explosion that happened at a hydrogen refueling station in Kjørbo, Norway (June 2019); for the simulation of this case, they used a particular CFD software called FLACS[2]. They showed that the reason behind the explosion was a failure in the assembly of the high-pressure hydrogen reservoir of this station in which a low torque bolt led to hydrogen leakage and after around 4 seconds an ignition occurred, probably because of an electrostatic discharge [6], [11].

The following numerical studies which were conducted in the past decades are those which tried to simulate high-pressure hydrogen release in an unconfined environment. Therefore, they did not consider the effect of existing obstacles in the domain, and their only focus was to investigate the hydrogen jet condition and the probable ignition in the immediate vicinity of the leakage.

The first one is a study by Liu et al. in which they tried to use a direct numerical method to investigate the spontaneous ignition of a high-pressure hydrogen release into the air; the point is they utilized a NON-MUSCL TVD[3] scheme for convective fluxes of the flow. Also, they used a detailed chemical kinetics mechanism with 18 elementary reactions and 9 species for capturing the onset of the possible combustion. Besides, it is claimed that for predicting a correct temperature in the 2D-simulation, they had to use a uniform rectangular mesh size of 10 μm. They repeated the simulation for 3 different initial hydrogen pressure (10, 40, and 70 MPa) and a small leakage with the diameter of 1 mm. Their main achievement was that although the shock-induced ignition occurs at the front region of the contact surface in 40 and

[1] https://www.gexcon.com/

[2] Flame Acceleration Simulator (FLACS) is a commercial CFD software developed by Gexcon which is particularly used in the field of gas explosion and dispersion [10].

[3] The Monotonic Upstream-centered Scheme for Conservation Laws (MUSCL) scheme, and Total Variation Diminishing (TVD) schemes will be discussed in the next chapter.

70 MPa cases, the local combustion is quenched due to forming the H_2O molecules (which reduce the temperature) and the unsteady nature of the flow in that region [12].

The second work is a project done by Xu et al. in which they utilized a modified version of KIVA-3V[1] CFD software. Their main approach was to apply a "*mixture-averaged multi-component*" method, that has higher-order numerical schemes in comparison with the original KIVA-3V, for solving the species transport equation of ideal gases by overcoming the false numerical diffusion; along with using a detailed chemical reaction mechanism with 21 steps and 8 species. The mentioned numerical scheme is based on the Arbitrary Lagrangian-Eulerian (ALE) method in which in the first step (Lagrangian phase), there is no convective flux across the cell boundaries, and cell vertices move with the fluid flow; in the second step (rezone phase), the flow field is considered frozen and the cell vertices move to a new computational mesh. Their general scenario was a release of pressurized hydrogen through a tube, initially filled with stagnant air, into an open space [8]. Their relevant case studies are tabulated in the following:

Table 1.1: Computational Parameters of the case studies [8]

Parameters	Case 1	Case 2	Case 3
Release Pressure ($\boldsymbol{bar}$)	40	70	70
Tube Diameter ($\boldsymbol{cm}$)	1	1	1
Length of Tube ($\boldsymbol{cm}$)	2	2	6
Diameter of Simulated Open Space ($\boldsymbol{cm}$)	16	16	16
Length of Simulated Open Space ($\boldsymbol{cm}$)	10	10	10
Number of Grids	1,600,000	800,000	
Minimum Grid Spacing ($\boldsymbol{\mu m}$)	20	40	

They tried to investigate the effects of the release pressure and also the length of the tube on occurring self-ignition. Based on their results, with a sufficient tube length and initial pressure (in this study 6 cm and 70 bar), the auto-ignition takes place inside the tube. In *Case 1*, there was no auto-ignition. However, in *Case 2*, a self-ignition initially occurs at the center of the contact surface because of the turbulence-enhanced mixing, but by decreasing the temperature due to dilution of the contact surface mixture, as well as the flow divergence, the flame is quenched [8].

Another research that is going to be reviewed is a project carried out by Maxwell and Radulescu. With referring to the Liu et al. and Xu et al. work, they began the project with the claim that numerical studies which had been conducted until that time on the subject of high-pressure hydrogen release, did not have sufficient resolution, i.e. necessary solver accuracy to capture spontaneous ignition. For overcoming this issue, they presented a model consisting of several independent, approximate steps written in the C++ language in association with using the Cantera libraries for evaluating thermodynamic properties. In this method, the ignition of a highly reactive homogenous hydrogen-air mixture is taking into account along with a realistic expansion rate derived from the shock-tube problem. They repeated the simulations for the

[1] The KIVA family software, developed by Los Alamos National Laboratory, is a commercial CFD software basically focused on simulating complex flow and combustion processes in internal combustion engines. https://www.lanl.gov/projects/feynman-center/deploying-innovation/intellectual-property/software-tools/kiva/index.php

initial hydrogen-air pressure ratios from 75 to 800 and based on their results, as shown in *Figure 1.4* with a dashed line, by increasing the pressure ratio the critical hole size decreases; also, they indicated that the pressure ratio controls the shock strength while the hole size controls the hydrogen expansion rate [13].

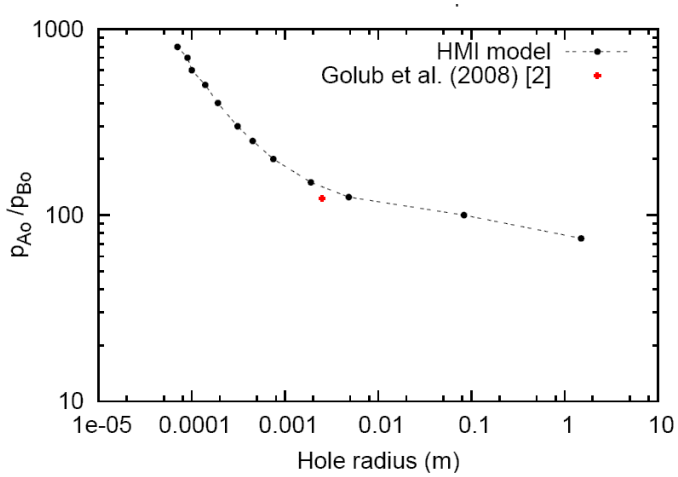


Figure 1.4: Critical hole sizes[1] for various pressure ration (initial hydrogen pressure/ambient air pressure) [13]

As the last review of this section, a paper by Bragin and Molkov is going to be studied here. Their intention was to propose a numerical model for simulating the release of high-pressure hydrogen that can achieve the same results as the conducted experimental studies; for this purpose, they compared their results mainly with two studies done by Mogi et al. (2008) and Golub et al. (2007). An LES[2] modeling concentrating on simulating vortices as accurate as possible along with a 21-step chemical reaction mechanism with 37 elementary reactions was the method used by them in FLUENT 6.3.26 CFD software. In general, this study can be divided into 2 sections, the first one is the spontaneous ignition in tubes which has approximately the same results as the discussed work done by Xu et al.; while the second one is the transition of this spontaneous ignition into a sustained jet fire out of the tubes which despite the Xu et al. study, they have shown that can be achieved. The initial hydrogen pressure for the latter part was 145 bar which was released in a tube with the length and width equal to 185 and 4 mm, respectively. The remarkable point is that in this research the grid resolution is much coarser than the other studies in this section; even after two steps of grid refinement, the cell sizes were in the order of hundreds of micrometers. Eventually, they concluded that the key point to this transition is the initial jet formation close to the tube exit in where formed vortices separate the flame jet into upstream and downstream combusting regions, and while the downstream one is blown away, the upstream region is stabilized [14].

1.2.3 Combined Studies

The last category of studies is a combination of the previous two ones in which it is tried to validate the results of each part with another one by carrying out the same case scenario experimentally or numerically.

[1] Critical hole size is the hole size below which the ignition is quenched.

[2] Large-Eddy Simulation

Among the best instances of this category, there are studies done by the same group of researchers known as Golub et al. which were conducted mostly in the first decade of the 21st century. The scenario of all these investigations was releasing of pressurized hydrogen through a tube into an oxidizer. The main focus, also, was to find a relation between the pressure ratio (P_{H_2}/P_Air), the temperature of gases, the nozzle diameter, and the tube length.

In the earlier studies, they emphasized the importance of the effect of initial temperature on hydrogen shock-induced ignition. Based on that, with hydrogen and air initial temperature of 300 K, for nozzle diameters more than 3 mm and the pressure ratio between 150 and 400 the ignition takes place, but for the sizes equal or less than 2.6 mm the combustion is quenched. By increasing the initial temperature to 400 K, even for a 2 mm nozzle and 200 pressure ratio, the ignition occurs. For the simulations of this study, they used the mentioned ALE algorithm together with a chemical kinetics model including 9 elementary reactions in FORTRAN-90. Also, it is important to note that the grid resolution of these cases was between $10-40\ \mu m$ [15], [16].

In the later studies, they emphasized the importance of the tube length effect. As discussed also in other studies, the tube length and the required release pressure to spontaneously ignite hydrogen, are inversely related to each other; the more the tube length, the less the required pressure.

Table 1.2: Summary of the research experimental results (ignition occurrence inside the tube is marked with Yes and No) [17]

Tube	Pressure [atm]	Distance Between Diaphragm And Pressure Transducer [mm]			
Rectangular		43	93	143	
	29	No	No	No	
	34	No	No	Yes	
	40	No	Yes	Yes	
	54	Yes	Yes	Yes	
Cylindrical		33	47	90	135
	38	No	No	No	No
	40	No	No	No	Yes
	52	No	No	Yes	Yes
	80	No	Yes	Yes	Yes
	96	Yes	Yes	Yes	Yes

All experiments and simulations are done for a tube cross-sectional area equal to 20 mm^2 which means the diameter of the cylindrical tube and the length of the rectangular tube are equal to 5.05 and 4.47 mm, respectively. As seen in *Table 1.2*, the spontaneous ignition at the same distance from the burst disk occurs in the rectangular tube at lower pressures than the cylindrical one. The other major achievements of this study regarding the pressure ratio are that below 96 atm for the cylindrical tube and 54 atm for the rectangular one, no ignition will take place inside a tube with 33 and 43 mm length, respectively. In addition to this, below 40 atm for the cylindrical tube and 34 atm for the rectangular one, no ignition will occur inside a tube even as long as 135 and 143 mm, respectively. At last it should be mentioned that their numerical results showed acceptable agreement with the experimental results [17], [18].

1.3 Defining the Present Project Scenario

Almost in all reviewed studies in this chapter, spontaneous ignition of high-pressure hydrogen was investigated either inside and outside a release tube or outside a high-pressure chamber, close to the burst disk (diaphragm). Despite all these, the downstream flow field conditions far from the release point which is affected by the shock waves, and also the effects of existing obstacles and subsequently reflected shock waves and rarefaction waves in this area are what this project is focused on. On the other hand, the desired hydrogen pressure for this project, $70\ MPa$, is determined by the maximum conventional pressure that is used for hydrogen storage, especially in transportation (*Table 1.3*).

Table 1.3: Different types of hydrogen compressed tank [19]

Cylinder Types	Materials	Features	Applications	Hydrogen storage pressure and mass percent ($WT\%$)
Type I	All metal	Heavy Internal corrosion	For industrial, not suited for vehicular use	17.5-20 MPa 1 $WT\%$
Type II	Metal liner with hoop wrapping	Heavy Short life due to internal corrosion	Not Suited for vehicular use	26.3-30 Mpa
Type III	Metal liner with full composite wrapping	Lightness High burst pressure No permeation Galvanic corrosion between liner and fiber	Suited for vehicular use 25-75% mass gain over I and II	35 MPa: 3.9 $WT\%$ -70 MPa: 5 $WT\%$
Type IV	Plastic liner with full composite wrapping	Lightness Lower burst pressure Permeation through liner High durability against repeated charging Simple manufacturability	Longer life than Type III (no creep fatigue)	70 MPa: more than 5 $WT\%$

According to all these, the proposed scenario for this project is a release of $70 - MPa$ hydrogen into the atmospheric air in a semi-confined environment with an existing obstacle right ahead of the hydrogen jet. The detailed geometry and the reasons for choosing it will be discussed in the next chapter.

2 CFD Background

In this chapter, it is tried to address the Riemann problem topic which was mentioned in section *1.2.2 Numerical Studies*. For this purpose, at first, the fundamental physical and numerical reasons behind this issue will be discussed. After that, the main approach for solving this problem and finally the relevant Riemann solvers to this project will be investigated.

2.1 Conservation Laws and Characteristic Curves

Physical conservation laws in the context of mathematics can be written in 3 different forms. (I) Integral form, which is written based on a control volume (Ω) and the fluxes through its boundaries ($\partial\Omega$). (II) Conservative form, in which the divergence theorem is applied to the integral form to convert the surface integrals into volume integrals. (III) Primitive form, in which the chain rule is exerted on the divergence of fluxes in the conservative form. Each of these has its special advantages which makes it useful for a specific kind of numerical investigation of physical phenomena. As a simple example, a scalar one-dimensional conservation law is considered below. In this equation, u is the conserved variable which is a function of time and space $u(t,x)$, S is the source term, and f is the flux. This equation can be written as [20], [21]:

$$\begin{cases} (Integral\ Form): & \dfrac{d}{dt}\displaystyle\int_{\Omega} u\,dx + \int_{\partial\Omega} \vec{f}(u).\vec{n}\,ds = \int_{\Omega} S(u)\,ds \\ (Conservative\ Form): & \dfrac{\partial u}{\partial t} + \nabla\cdot\vec{f}(u) = S(u) \\ (Primitive\ Form): & \dfrac{\partial u}{\partial t} + \dfrac{d\vec{f}}{du}\cdot\nabla u = S(u) \end{cases} \tag{2.1}$$

Characteristic curves or briefly characteristics are the curves (in the above case, they are lines) along which the PDEs in conservation laws become ODE[1]s, i.e. the dependent variable(s) becomes only a function of time. As an example, in the above *Equations (2.1)*, if straight lines in the form of $x = a\cdot t + x_0$ in the $t-x$ diagram, depicted in *Figure 2.1*, are taken into account, the dependent variable u becomes a function of time $u\big(t,x(t)\big)$ along these lines. So, the partial derivative $\partial u/\partial t$ in *(2.1)* changes into the derivative du/dt. For $a = d\vec{f}/du$[2] this derivative on the specific lines can be calculated based on the chain rule as follows [20], [21]:

$$\begin{cases} \dfrac{du}{dt} = \dfrac{\partial u}{\partial t} = \dfrac{\partial u(t,x)}{\partial t} = \dfrac{\partial u}{\partial t}\times\dfrac{\partial t}{\partial t} + \dfrac{\partial u}{\partial x}\times\dfrac{\partial x}{\partial t} = \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} \xrightarrow{a=d\vec{f}/du} \dfrac{du}{dt} = \dfrac{\partial u}{\partial t} + \dfrac{d\vec{f}}{du}\cdot\dfrac{\partial u}{\partial x} \\ (Primative\ Form): \dfrac{\partial u}{\partial t} + \dfrac{d\vec{f}}{du}\cdot\nabla u = S(u) \rightarrow \dfrac{\partial u}{\partial t} + \dfrac{d\vec{f}}{du}\cdot\dfrac{\partial u}{\partial x} = S(u) \end{cases}$$
$$\xrightarrow{yields} \frac{du}{dt} = S(u) \tag{2.2}$$

[1] Ordinary Differential Equation

[2] This slope (a) is also called the characteristic speed

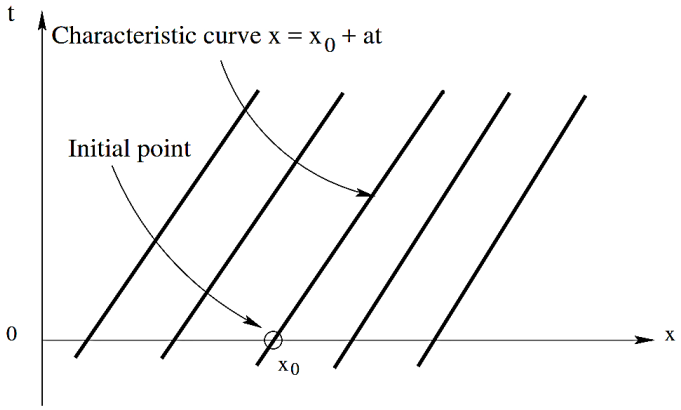
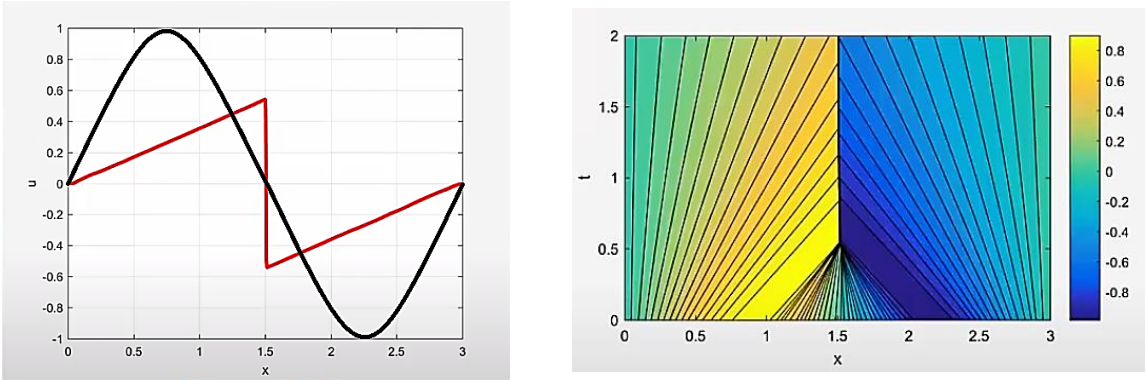


Figure 2.1: Characteristic lines for the mentioned conservation law with $a > 0$ and initial conditions: $t(0) = 0$, and $x(0) = x_0$ [21]

The interpretation of *Equation (2.2)* is that we can analytically predict the behavior of the solutions in the continuous regions. For instance, in a very simple case if the source term $S(u) = 0$ which makes the derivative $\frac{du}{dt} = 0$, then the solution u will be constant along all the characteristics. And as a result, given an initial profile $u(t(0), x) = u(0, x) = u_0(x)$ through the entire domain, the above conservation law *(2.1)* will just transfer this profile according to the a. For $a > 0$ the profile will be translated to the right, for $a < 0$ it will be translated to the left, and for $a = 0$ it remains at the same point. The Burgers equation is a perfect example for this case in which the flux term $f(u) = u^2/2$, in addition to all the previous assumptions.

$$(Burgers\ Equation):\ \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \tag{2.3}$$

For such a case, if the initial profile $u_0(x)$ is considered as the black curve in *Figure 2.2 (a)*, the developed solution (the red curve) is simply a translation of the initial profile based on the slopes of the characteristics (a which is equal to $\frac{d\vec{f}}{du} = u$) [20], [21].



(a) (b)

Figure 2.2: (a) The $u - x$ Diagram: the black curve is the initial profile $u_0(x)$, and the red one is the developed solution after 2 seconds (b) The $t - x$ Diagram: the contour is showing the solution $u(t, x)$, the black straight lines are the characteristics [20]

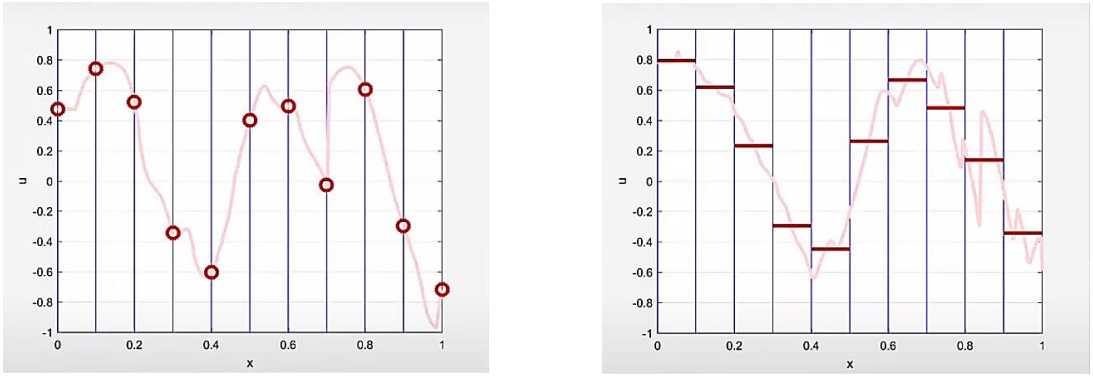
2.2 The Finite Volume Method (FVM)

For continuous (smooth) regions in the domain, like what was discussed in the previous sections, even the simple Finite Difference Methods (FDM), which are basically the Taylor series approximations, are working for numerical investigation of conservation laws. But, if there are discontinuities in the domain, like shock waves, then the spatial and time derivatives of u will not exist at those spots. In other words, the order of their approximation (the Taylor series approximation of spatial derivatives) will go to infinity. Thus, the last two forms of conservation laws, conservative form and primitive form, will not be applicable anymore. As the result, only the integral form is still valid and can be utilized for studying such cases. This is the milestone that numerical approaches must be changed from FDMs to FVMs. It is noteworthy that every FV scheme is not suitable for this kind of problems.

As shown in *Figure 2.3*, in finite volume methods, the computational domain is divided into a finite number of control volumes. The properties of each volume (cell) are stored as volume-averaged information in each cell ($\bar{u}_i$ for the $i-th$ cell) and then the solutions will be reconstructed at boundaries (reconstructed solution $\tilde{u}(\bar{u}_i, \bar{u}_{i+1})$ which is the solution at $u_{i+1/2}$).

$$\frac{d}{dt}\int_{(i)\Delta x}^{(i+1)\Delta x} u\,dx + f(u)\Big|_{(i)\Delta x}^{(i+1)\Delta x} = 0 \xrightarrow{yields} \frac{d}{dt}(\bar{u}_i \cdot \Delta x) = f\left(u_{i-\frac{1}{2}}\right) - f\left(u_{i+\frac{1}{2}}\right) \qquad (2.4)$$

This approach is the same between all FV schemes, but what is different between them, as shown in *Equation (2.4)*, is the method used to approximate the reconstructed solutions ($u_{i\mp1/2}$) as a function of averaged stored values ($\bar{u}_i$). Also, it is worth knowing that there are some FV schemes in which instead of trying to first reconstruct the solutions ($u_{i\mp1/2}$) and then compute the fluxes ($\vec{f}(u_{i\mp1/2})$), they try to directly approximate numerical fluxes ($f_{i\mp1/2}$) [20].



(a) (b)

Figure 2.3: (a) Finite difference approach vs. (b) Finite volume approach of storing properties in each cell [20]

2.3 Non-Uniqueness of Solutions and the Entropy Condition

As mentioned in the previous section, every FV scheme is not appropriate for dealing with sharp discontinuities. For example, using the simple Central Flux scheme ($u_{i+1/2} = \frac{\bar{u}_i + \bar{u}_{i+1}}{2}$) for the Burgers equation with the below initial profile (*Figure 2.4*) will create oscillation at the shock wave. The reason behind this behavior is that the volume-averaged solution at the cell

composed of shock wave ($\bar{u}_i$) which is demonstrated by a red line, cannot satisfy the flux equation. According to the flux term of the Burgers equation $f(u) = u^2/2$, for these initial conditions flux will be $f(u) = 1/2$ in the entire domain, but the approximated flux in the $i-th$ cell will be $f(u) = \bar{u}_i^2/2$ which cannot be equal to $1/2$ with utilizing the Central Flux scheme.

There are simple adjustments for this situation, like the one used in the Upwind scheme. Regarding this scheme, for capturing the correct behavior of the shock wave, the direction in which the solution is bumping into the shock wave should take into account to approximate a correct value for the flux. Based on that, to reconstruct the solution at $u_{i-1/2}$ and consequently approximate a correct flux at this interface, because $a = df/du = u > 0$, the left value of the shock wave cell should be considered, and in the same way for the solution at $u_{i+1/2}$, the right value [20].

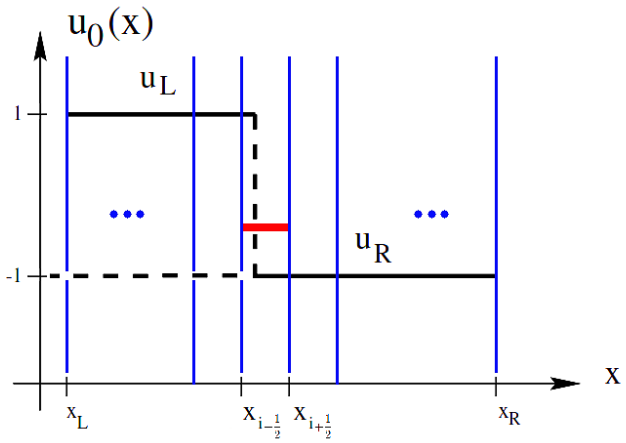


Figure 2.4: Initial profile $u_0(x)$ of the Burgers equation

Unfortunately, this kind of simple solution would not be always useful for large gradients and that is where the Riemann problem and subsequently Riemann solvers come into play. For instance, consider the below initial profile (*Figure 2.5*) for the Burgers equation which is again a discontinuity in the domain. But in this case, in contrast to the previous one, the solutions (information) are emanating from the shock wave, not running into the shock wave.

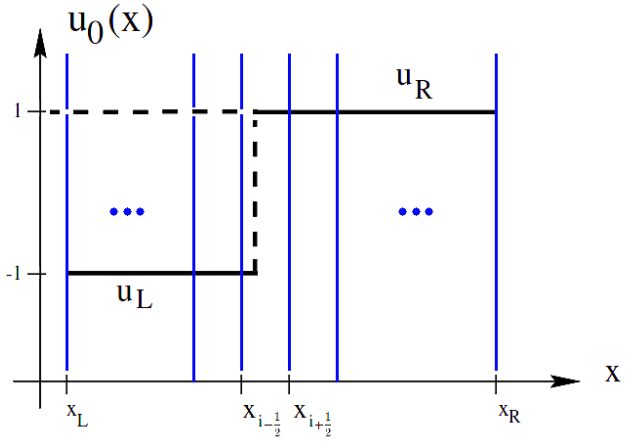
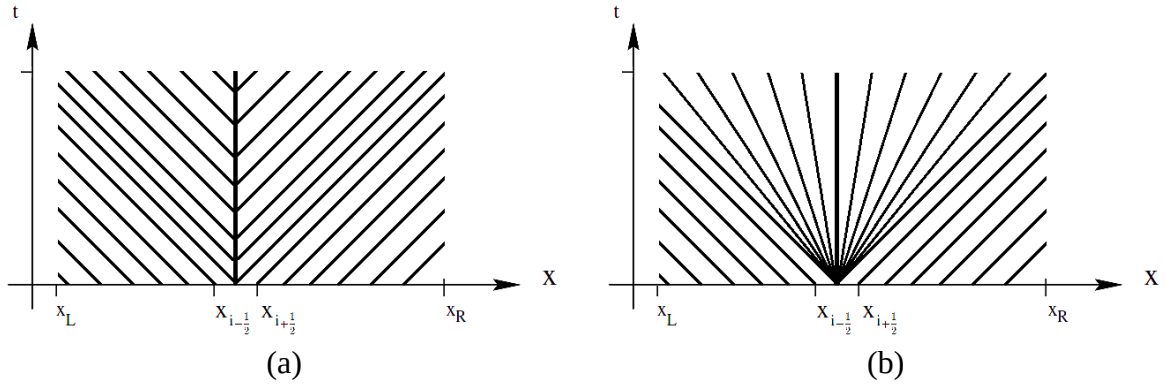


Figure 2.5: Initial profile $u_0(x)$ of the Burgers equation

The issue of such a case is that there is more than one solution for this problem based on the discussed schemes. However, among these solutions, there is only one physical possibility which is the aim of the Riemann solvers to find. If these solutions are drawn in the $t-x$ diagram, there are two main types of them. The first one is the stationary type which an example of those is shown in *Figure 2.6 (a)*. Regarding this type, there is a stationary discontinuity from which characteristics emanating, i.e. the shock wave is creating information in the domain. This category of solutions is not physical because the entropy of the system is decreasing. In other words, the dissipation of the shock wave is negative. The other type which is a unique solution, as indicated in *Figure 2.6 (b)*, is what is called expansion fan. This solution is a moving one which means after a while the steep slope of the discontinuity, which is infinite at the initial time, starts to flatten out. This one is the only physical solution to this problem because, despite the stationary solutions, the expansion fan conserves the entropy in the system. So, the limiter of the Riemann solvers is what is called the entropy condition. Analyzing the entropy of a possible solution is the criterion that determines whether a solution is physical or non-physical [20], [21].



(a) (b)

Figure 2.6: Characteristic lines of (a) Non-physical and (b) Physical solutions of the Burger equation with the initial conditions shown in *Figure 2.5*

One of the earliest numerical approaches for dealing with this problem was the entropy satisfying first-order Godunov numerical flux which guaranteed to have a physical solution. The Godunov scheme is basically a linear piecewise flux function. Based on that, for example for solving the Burgers equation mentioned above, the condition around the discontinuity is divided into 4 general cases for reconstructing the flux at the interface [20]:

$$\begin{cases} u_L \geq 0\,, u_R \geq 0: & f_{i+1/2} = f(u_L) \\ u_L < 0\,, u_R < 0: & f_{i+1/2} = f(u_R) \\ u_L < 0\,, u_R \geq 0: & f_{i+1/2} = 0 \\ u_L \geq 0\,, u_R < 0: & \begin{cases} u_L + u_R \geq 0: & f_{i+1/2} = f(u_L) \\ u_L + u_R < 0: & f_{i+1/2} = f(u_R) \end{cases} \end{cases} \tag{2.5}$$

2.4 Relevant Riemann Solvers to This Project

During the past century and even still today, many efforts have been made to propose methods for solving the Riemann problem. In the following, some of these methods which are relevant to this project will be studied.

2.4.1 The TVD Schemes

Numerical schemes that use the total variation diminishing (TVD) capability are called TVD schemes. The TVD property of these schemes assures the numerical solution of avoiding spurious oscillations near the sharp discontinuities with preserving monotonicity. And this is achieved by diminishing the total variation of the conserved variable(s). Again if the Burgers equation is considered as an example, the total variation of u will be defined as *Equation (2.6)* [22], [23].

$$TV\big(u(t)\big) = \sum_i |u_{i+1}(t) - u_i(t)| \tag{2.6}$$

Regarding the TVD method, this total variation of the solution must not increase:

$$TV\big(u(t+\Delta t)\big) \leq TV\big(u(t)\big) \tag{2.7}$$

In *Equation (2.7)*, Δt is a time step of the time discretization. It is proved that the TVD capability and monotonicity preserving are corresponding to each other. Monotonicity preserving of a numerical scheme means that if $u(t)$ is a monotonic function of space, then $u(t+\Delta t)$ will follow the same behavior [21], [23]. The Riemann solvers that will be discussed in the following are TVD schemes.

2.4.2 The FLIC Scheme

The Flux-Limiter Centered (FLIC) scheme, which will be used in the USN[1]_FLIC code simulation in the next chapter for coping with the Riemann problem, is a TVD second-order scheme introduced by Eleuterio F. Toro in 1999. It is a combination of "*the 1st order accurate FORCE scheme and the 2nd order Richtmyer version of the Lax-Wendroff scheme*". In this method, it is tried to calculate the interface fluxes ($f_{i+1/2}$) based on an equation in which the order of approximation can be changed according to the smoothness of the flow; For the regions with sharp discontinuities the order of the scheme will be one, while for the smooth regions it will tend to two [22]. This modifying is done by what is called flux limiter ($\emptyset$). Flux limiters are calculated with different methods based on the gradients of the flow, and their responsibility is to make sure that the slopes of the piecewise approximations do not exceed a certain limit to secure the TVD solution.

2.4.3 The MUSCL Scheme

The Monotone Upstream–centered Scheme for Conservation Laws, or briefly the MUSCL scheme, which is also known as the Variable Extrapolation approach, was first introduced by Bram van Leer in 1979. The main idea behind that was to change the piecewise constant approximation of the first-order Godunov scheme by extrapolating the solutions at the interfaces to achieve a higher order of accuracy. Among the MUSCL schemes, there are several different ways for that extrapolation [21], [24]. In this thesis, the Kurganov and Tadmor central MUSCL schemes are those which will be used in the OpenFOAM simulation in *Chapter 5*.

[1] University of South-Eastern Norway

In this method, first the right and left extrapolated solutions at the interfaces ($u^R_{j\mp1/2}$ and $u^L_{j\mp1/2}$ in *Figure 2.7* on the next page) will be calculated based on the averaged cell solutions ($\bar{u}_i$) and flux limiters. After that these extrapolated values will be put in a pair of equations to calculate the interface fluxes. These equations are a function of extrapolated solutions and the characteristic speeds at the interfaces ($a_{i\mp1/2}$) which will be used in conservation equations like *Equation (2.4)* [24].

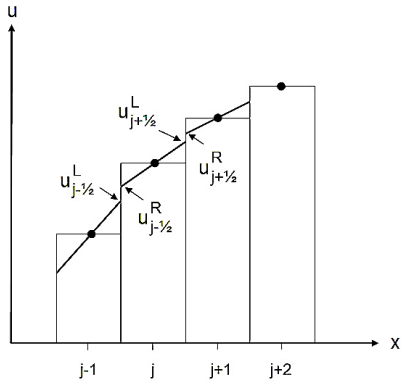


Figure 2.7: Piecewise linear extrapolation of solutions in a MUSCL scheme [24]

3 USN-FLIC Code Simulations

This chapter is about the USN-FLIC code and the procedure that has been undergone to simulate the defined scenario with it. In the beginning, the CFD theory of the code will be briefly investigated and after clarifying the simulations' settings, the results and the required next steps will be discussed.

3.1 The Code Properties and Governing Equations

The USN-FLIC code, initially, was written in MATLAB to simulate gas explosions by using the discussed numerical FLIC scheme. The original code includes a combustion model in which a combined total reaction rate is used. This total reaction rate is the maximum reaction rate from chemical kinetics and from the mixing rate (turbulent combustion rates). And actually, it shows which reaction rate is dominant in each stage of the explosion [22]. The code can also perform the simulation in cylindrical coordinates. But the provided version of the USN-FLIC code for this project can only simulate two-dimensional CFD problems in a Cartesian coordinate without considering the heat releasing reaction. Thus, in fact, the provided code just studies the first step of the chemical reaction in which the reactants react to radicals without any heat release.

The main steps of the utilized USN-FLIC code are as follows:

1. Setting the initial conditions
2. Creating the geometry of the computational domain
3. Calculating the domain properties in a for-loop for a desired number of iterations
   3.1. Calculating the time step based on the Courant-Friedrich-Levi (CFL) number equal to 0.1 for the first loop and 0.9 for others (according to the CFL number criterion, no wave should travel more than the characteristic length of a control volume)
   3.2. Calculating the induction time τ for a stoichiometric hydrogen-air mixture based on the following equation:

$$\tau = A_\alpha \left(\frac{T}{p}\right) \mathrm{e}^{\left[-B_\alpha + \frac{C_\alpha}{T} + D_\alpha \left(\frac{p}{p_{atm}}\right)^2 e^{\left(\frac{E_{a_\alpha}}{T}\right)}\right]} \tag{3.1}$$

   For $A_\alpha = 6.2335e+10\ [\frac{Pa}{K}.s]$, $B_\alpha = 35.1715$, $C_\alpha = 8530.6\ [K]$, $D_\alpha = 7.22e-11$, $E_{a_\alpha} = 21205\ [K]$.

   3.3. Calculating α based on the following equation. This variable shows the stage of the gas explosion. As it tends to 1the two-step combustion model shift from the radical production step to the heat release step:

$$\alpha = \int 1/\tau \, dt \tag{3.2}$$

   3.4. Calculating the density and the hydrogen-air laminar burning velocity for a stoichiometric mixture at a constant pressure based on the following equation:

$$S_L = 2.38 \left(1 + 1.54 \log_{10}\left(\frac{p}{p_0}\right)\right) \left(\frac{T}{T_0}\right)^{0.43} \tag{3.3}$$

   For $p_0 = p$, and $T_0 = 291\ [K]$.

3.5. Calculating the hydrogen-air turbulent burning velocity based on the computed laminar burning velocity as:

$$S_T = S_L\left(1 + A\frac{\sqrt{Re \cdot Pr}}{Da^{0.25}}\right) \tag{3.4}$$

For $A = 0.52$, Reynolds number $Re = \frac{u' \cdot l}{\nu}$, Prandtl number $Pr = \frac{\nu}{\kappa}$, and Dahmkohler number $Da = \frac{\delta}{u' \cdot \tau_c}$.

3.6. Solving the conservation equations by applying the FLIC scheme on the y-direction variables' discretizations and updating all variables (conservation equations will be discussed in the next chapter)

3.7. Returning the value of the variables at "wall" boundaries to the initial settings

3.8. Again, calculating the turbulent burning velocity

3.9. Applying the FLIC scheme this time on the x-direction variables' discretizations and updating all variables

3.10. Repeating step *3.7*

The algorithm chart of the original USN-FLIC code is given in *Appendix C*. Furthermore, it is important to notice that the utilized equation of state in steps *3.6*, and *3.9* to model the internal energy is the ideal gas law. But in the original USN-FLIC code, in order to simulate the higher temperatures more accurately, this equation is used along with applying variable $\gamma = c_p/c_v$ value. For this purpose, the heat capacities are calculated based on temperature-dependent polynomial functions.

3.2 Simulation Setup

3.2.1 Geometry and Boundary Conditions

The geometry of this study, basically, was created according to the geometry of available apparatus in the combustion laboratory at USN to make further experimental investigations possible. The available system is a channel with a length, width, and height of 300, 10, and 10 cm, respectively, and a set of nozzles with different diameters. Based on the mesh independence test, which will be discussed in the next section, the grid resolution for simulating (2D) the defined scenario should be at least 100 μm. Creating a mesh with the specified dimensions and grid resolution will generate 30,000,000 cells that its analysis is beyond the power of the provided computational system. On the other hand, the utilized USN-FLIC code was not able to use the axisymmetric boundary condition to facilitate having a large geometry.

As the result of all these, a two-dimensional, semi-confined channel with the half size of the experimental one ($150 \times 5\ cm^2$) was chosen to simulate the case. The hydrogen reservoir boundary starts from the beginning of the channel up to 75 cm with a nozzle diameter of 0.15 cm. The reason behind this long reservoir selection was to reduce the initial pressure gradient at the output of the reservoir to avoid solver divergence. Moreover, the desired obstacle was created as a plane wall with a 2.5 cm height. The initial distance between the nozzle output and the obstacle was set to be 4 cm. The plan was to alter this distance to observe

the effect of that on the maximum temperature of the hydrogen-air mixture, although due to the results of the initial setup simulation, which will be given at the end of this chapter, this idea was given up. The schematic of the geometry is shown in *Figure 3.1*.

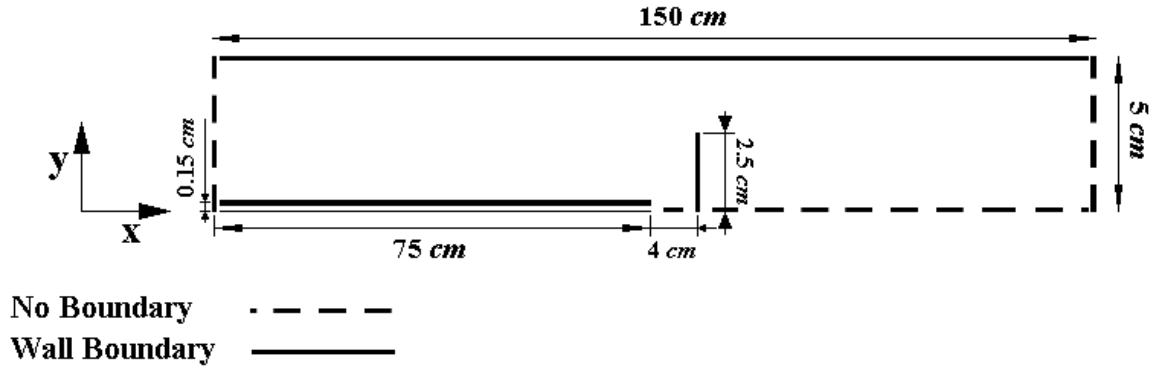
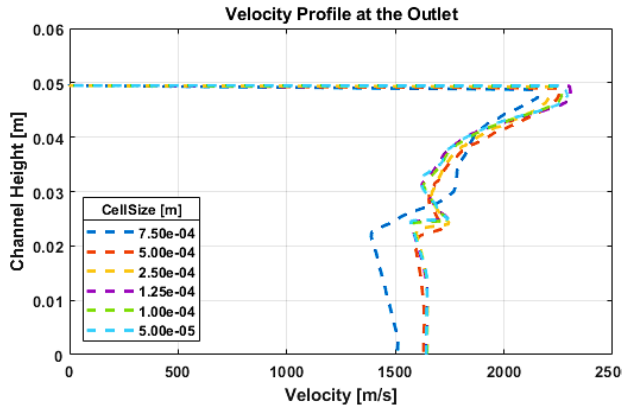


Figure 3.1: Sketch of the created geometry for USN-FLIC code simulation

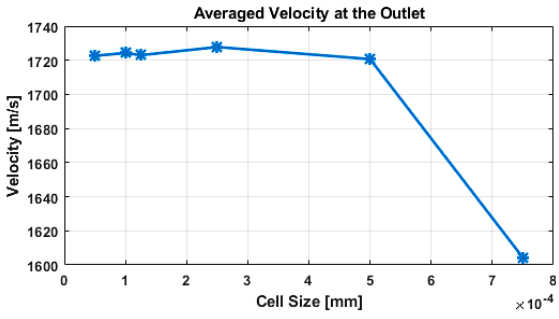
3.2.2 Mesh

As it was mentioned in the literature review section, a wide range of grid resolution from hundreds to tens of microns has been used for different numerical schemes and different goals. So, to find a proper mesh size without having experimental data for comparison, the best approach would be the mesh independence test. According to this method, some parameters from the results of the same simulations with different grid resolutions will be compared to each other to find a trend in which these parameters' values tend to specific numbers. This trend and the mesh size corresponding to these specific numbers determine the proper grid resolution for that solver and case. As common choices, usually the inlet and outlet velocity and pressure profiles along with the average of them are considered. In this study, in addition to the mentioned parameters the particle tracking function, which was written by the author for further studies of the USN_FLIC code, will also be used for this test. The core section of the particle tracking code is given in *Appendix D*.

The mesh independence test results are as follows:

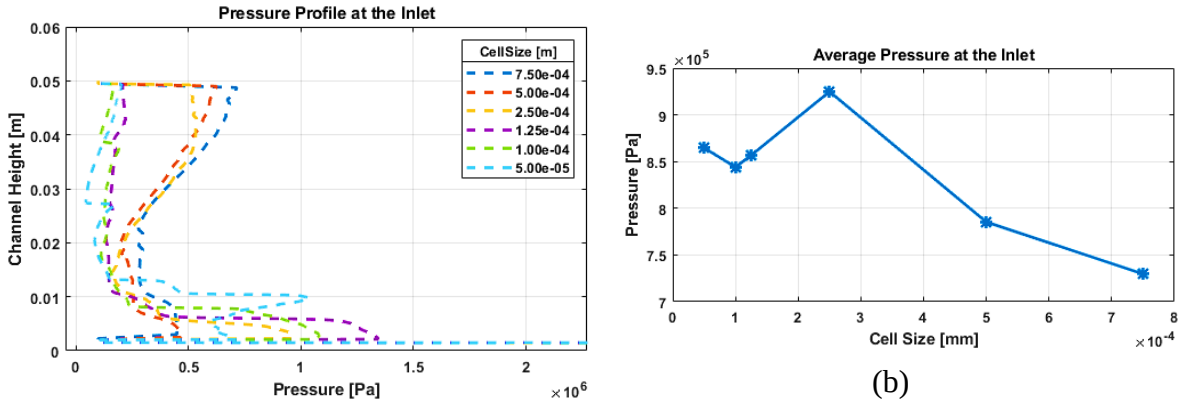


(a)



(b)

Figure 3.2: (a) Outlet velocity profile (b) Outlet averaged velocity for different grid resolutions from 50 μm up to 750 μm



(a) (b)

Figure 3.3: (a) Inlet pressure profile (b) Inlet averaged pressure for different grid resolutions from 50 μm up to 750 μm

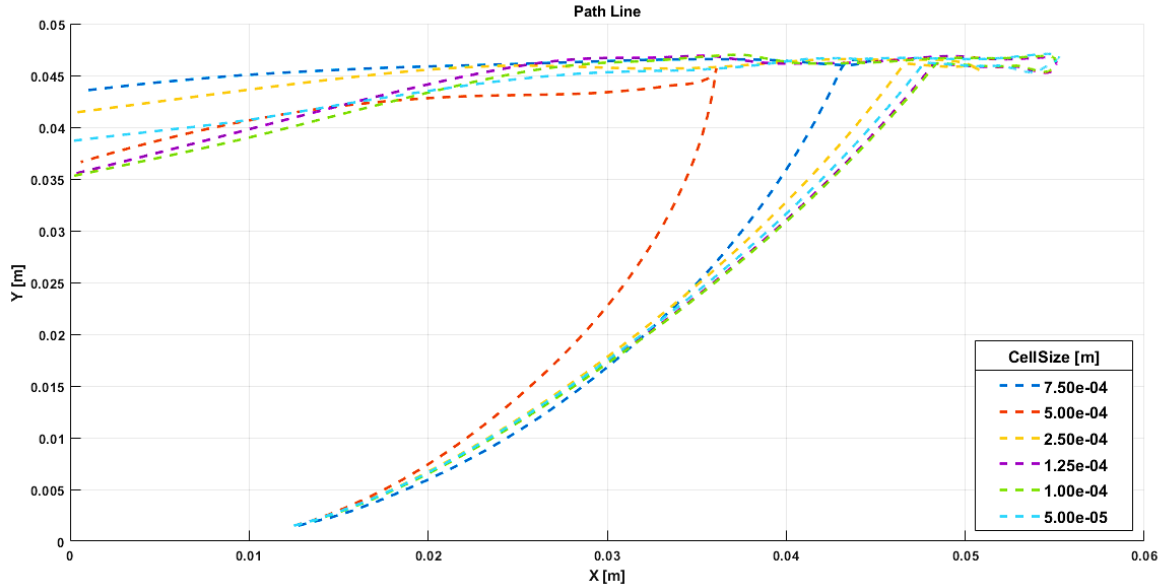


Figure 3.4: The path lines of a specific particle in different grid resolutions from 50 μm up to 750 μm

As can be seen in *Figure 3.2*, by increasing the grid resolution over 500 μm the outlet velocity has experienced a severe drop, however for higher resolutions the average outlet velocity values are approximately the same. On the other hand, for the inlet pressure shown in *Figure 3.3*, the averages have more fluctuated, although it can be detected that the inlet pressure profiles are divided into two main groups. First, the pressures of the 50, 100, and 125 μm resolutions. Second, the pressures of the 250, 500, and 750 μm resolutions. Furthermore, if the path lines of the specific particle at the inlet in *Figure 3.4* are considered, it can be seen that the path lines of the 100, and 125 μm are almost overlapped. With respect to all these, and other parameters' diagrams which are not brought in here (like inlet velocity or outlet pressure, or even density profile), the 100 μm grid resolution was chosen for the USN-FLIC code simulation. As the result, the total number of cells for a mesh with $dx = dy = 1e - 04$ will be almost 15000×500 which is 7,500,000.

3.2.3 Initial Conditions

The initial conditions that should be determined for this simulation are tabulated below. The initial density of the hydrogen reservoir is set based on the ideal gas law for initial pressure $p_{H_2} = 70\ [MPa]$, and initial temperature $T_{H_2} = 291\ [K]$. The reason for this choice is that the code is using the ideal gas equation of state. Therefore, using the real-gas density ($40\ kg/m^3$) at this pressure and temperature is interpreted a higher temperature by the solver. However, it is worth mentioning that another simulation based on the real density of hydrogen at this pressure and temperature ($r_{H_2} = 40\ [kg/m^3]$) was done, and the maximum temperature of this case was relatively $500\ K$ higher than the main case (just for more illustration the flow properties' diagrams at the same time as *Figure 3.6* is given at the end of this chapter in *Figure 3.10*).

Table 3.1: Initial conditions of the USN-FLIC code simulation

Air molecular weight $[kg/kmol]$	$\boldsymbol{mw}$	29e-03	**Burned molecular weight $[kg/kmol]$**	$\boldsymbol{mw_b}$	29e-03
Ambient initial pressure $[Pa]$	$\boldsymbol{p_0}$	1e+05	**Hydrogen reservoir initial pressure $[Pa]$**	$\boldsymbol{p_{H_2}}$	700e+05
Ambient initial temperature $[K]$	$\boldsymbol{T_0}$	291	**Hydrogen reservoir initial mass fraction**	$\boldsymbol{f_{H_2}}$	1
Ambient initial density $[kg/m^3]$	$\boldsymbol{r_0}$	$\frac{p_0 \cdot m_w}{8.314\ T_0}$	**Hydrogen reservoir initial density $[kg/m^3]$**	$\boldsymbol{r_{H_2}}$	57.86
Initial velocities $[m/s]$	$\boldsymbol{u_x, u_y}$	0	**Heat capacity ratios**	$\boldsymbol{\gamma_{air}, \gamma_{H_2}, \gamma_{burn}, \gamma_{unburn}}$	1.4

3.3 Simulation Results and Discussion

The temperature contours can be made from the simulation results (pressure, density, and hydrogen mass fraction) by using the following equation which is derived based on the ideal gas law:

$$T = \frac{p}{R \cdot \rho}\left(\frac{1}{\frac{f_{H_2}}{mw_{H_2}} + \frac{1 - f_{H_2}}{mw_{Air}}}\right) \tag{3.5}$$

In this equation, p and ρ are the pressure and density of a cell (mixture), and $R = 8.314\ [kJ/(kmol \cdot K)]$ is the universal gas constant. Hydrogen and air molecular weight are equal to $2e-03\ [kg/kmol]$, and $29e-03\ [kg/kmol]$, respectively.

As demonstrated in *Figure 3.5*, the temperature of the simulation reaches its maximum at $t = 8.1269e-05\ [s]$ and this has happened only in one cell which is determined by a red circle in this figure. Not only the value of this temperature is doubtful since such a high temperature ($> 5000\ [K]$) was not recorded in the previous studies, but also how this happened is not acceptable. At this time, the temperatures of only 5 cells are above $2500\ [K]$.

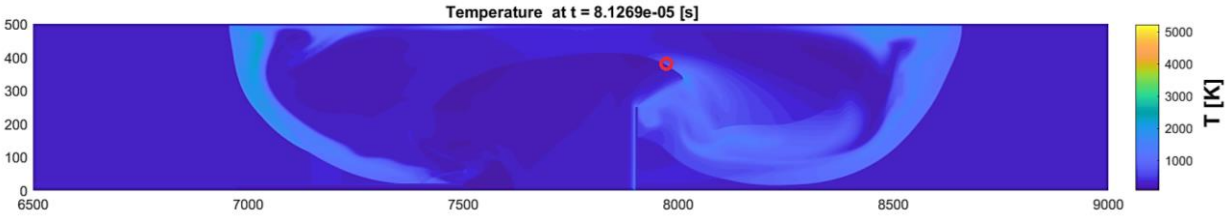


Figure 3.5: Temperature contour of the simulation at $t = 8.1269e - 05\ [s]$

If such these discontinuities are neglected and just the general trend of the flow is considered, it can be stated that the highest temperatures have happened several times in the order of $4000\ K$ and at the interactions between the shock waves and wall boundaries including the plane wall obstacle, the ceil of the channel, and the reservoir wall. An example of this phenomenon is illustrated in *Figure 3.6*. The highest temperature region is highlighted by a red circle in the temperature diagram, although this region is not important by itself because it can be seen that the corresponding area in the mass fraction diagram is pure air. So, there would not be any hydrogen spontaneous ignition hazard. However, this highest temperature can be considered as a criterion of shock waves intensity.

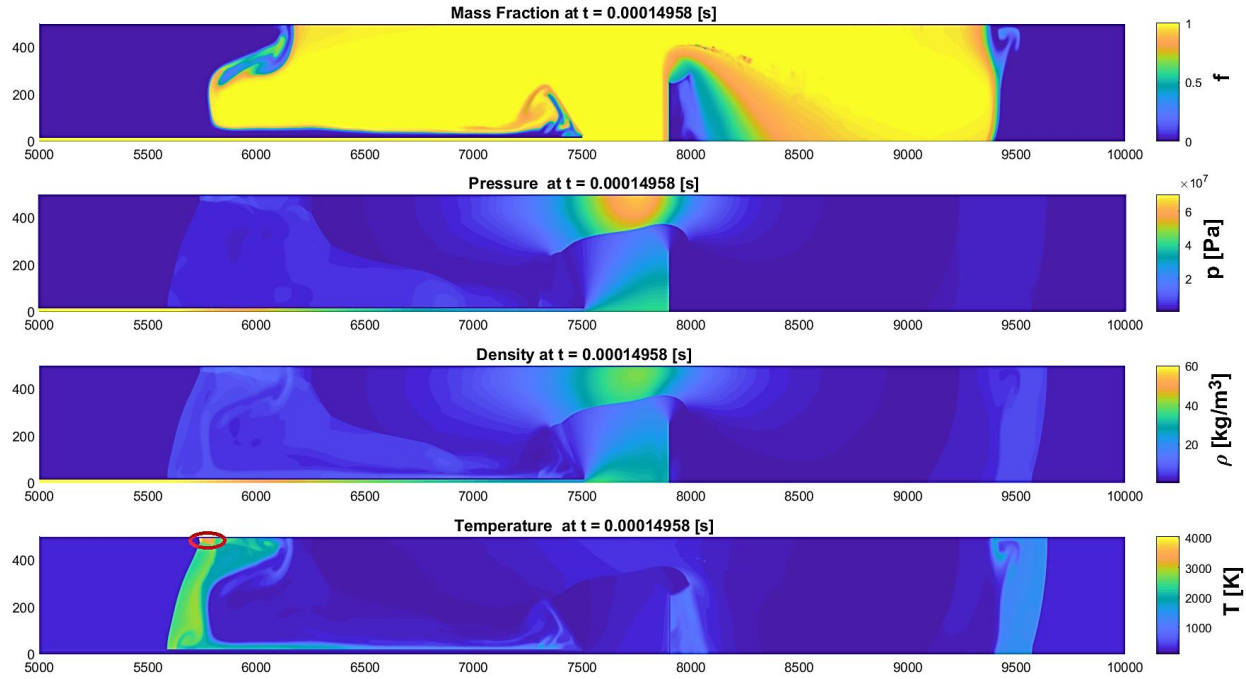


Figure 3.6: Flow properties of the simulation at $t = 1.4958e - 03\ [s]$

The regions of interest in this project are those in which the hydrogen content of the mixture is in the range of hydrogen flammability and the temperature is higher than the hydrogen auto-ignition temperature. As a rough approximation, if a threshold is applied to the temperature and pressure contours for $T > 858\ [K]$, and $0.05 < x_{H_2} < 0.75$, then the regions of interest are achieved (these numbers are based on the section *1.1* discussion). The volumetric hydrogen concentration is calculated based on *Equation (3.6)* which is derived from combining the mole

fraction and mass fraction equations and based on the fact that the molar and volumetric concentrations are corresponding to each other in the case of ideal gases.

$$x_{H_2} = \frac{1}{1 + \left(\left(\frac{1}{f} - 1\right)\frac{mw_{H_2}}{mw_{Air}}\right)} \tag{3.6}$$

As can be observed in *Figure 3.7*, these regions which are almost located near the wall boundaries, have a pressure range of 1 to 75 bar, and a temperature range of 858 to 2400 $[K]$. Even with raising the threshold, for example to $0.1 < x_{H_2} < 0.4$, still these regions persist with almost the same temperature and pressure ranges, although their thickness is reduced. Just as an example among the main three regions of *Figure 3.7*, a random cell located inside the red circle is picked. The flow properties at this point are $T = 1347\ [K]$, $p \approx 34\ [bar]$, and $x_{H_2} \approx 19.36\ [\%]$. If these conditions are set as the initial conditions of a zero-dimensional kinetics simulation in the Cantera toolbox, as shown in *Figure 3.8*, such a hydrogen-air mixture only needs 12 μs to ignite. And it is worth knowing that according to the USN-FLIC code simulation and the velocity of the flow at this spot and this time, the mixture has such induction time. Therefore, based on this simulation, even if the ignition does not happen or is blown away at the beginning of the high-pressure hydrogen release, there will be regions in the domain after a while, mainly near wall boundaries, which satisfy the initial conditions for a hydrogen spontaneous ignition. And it should be mentioned that this kind of self-ignition will be much more catastrophic than the initial ones, discussed in section *1.2*. Because the mixture cloud in this scenario is expanded much further and as the result, there is more combustible mixture in the domain.

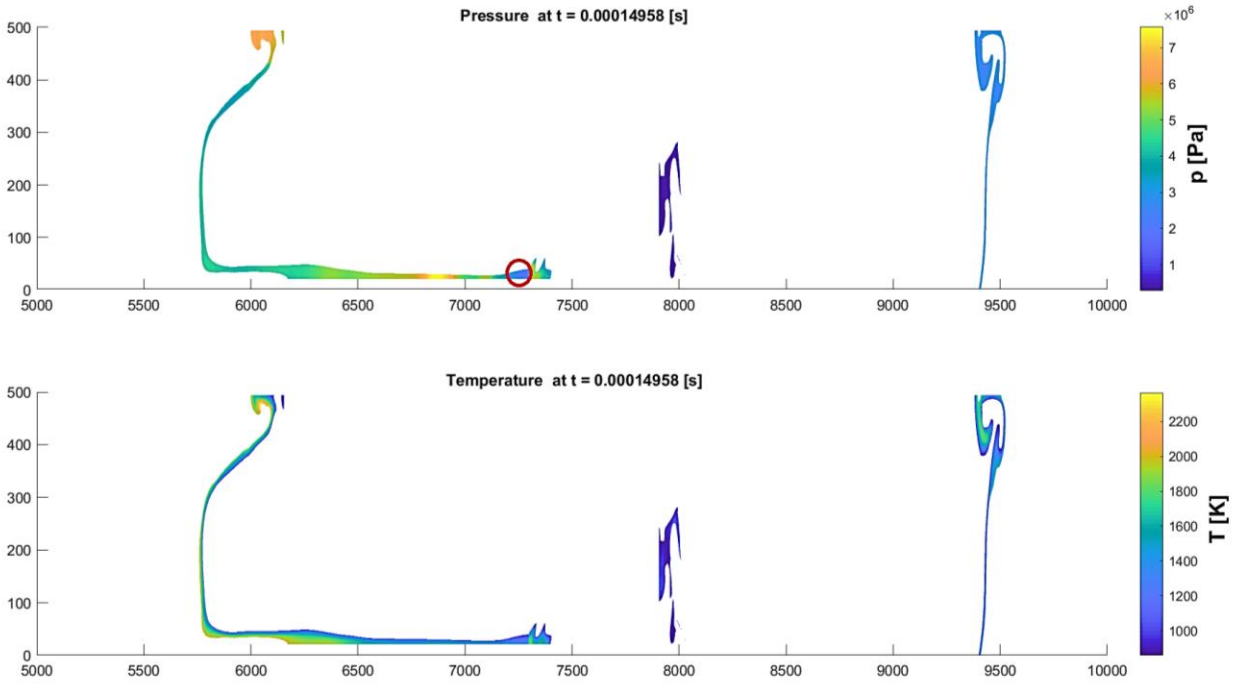


Figure 3.7: The regions of interest with temperatures higher than 858 $[K]$, and hydrogen concentration between 5% and 75% at $t = 1.4958e-04\ [s]$

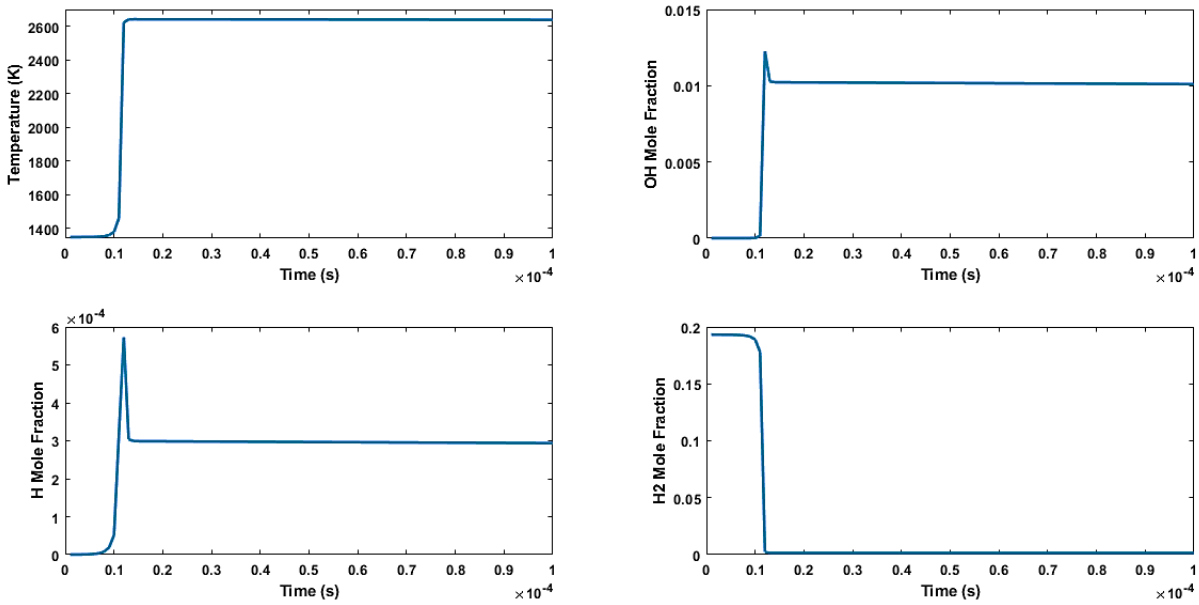


Figure 3.8: Cantera zero-dimensional kinetics simulation results of hydrogen-air combustion with initial conditions of $T_0 = 1347\ [K]$, $p_0 = 34.076e + 05\ [Pa]$, and $x_{H_2} = 19.357\ [\%\ in\ air]$

In addition to this scenario, as the flow propagates along the channel, it seems that there are regions in which air is trapped among the hydrogen cloud. At first glance, these areas seem to be potentially dangerous, but after calculating hydrogen concentrations for these regions, as demonstrated in *Figure 3.9*, the hydrogen content at these spots is much higher than the upper hydrogen flammability limit which makes them safe, at least in the investigated computational domain.

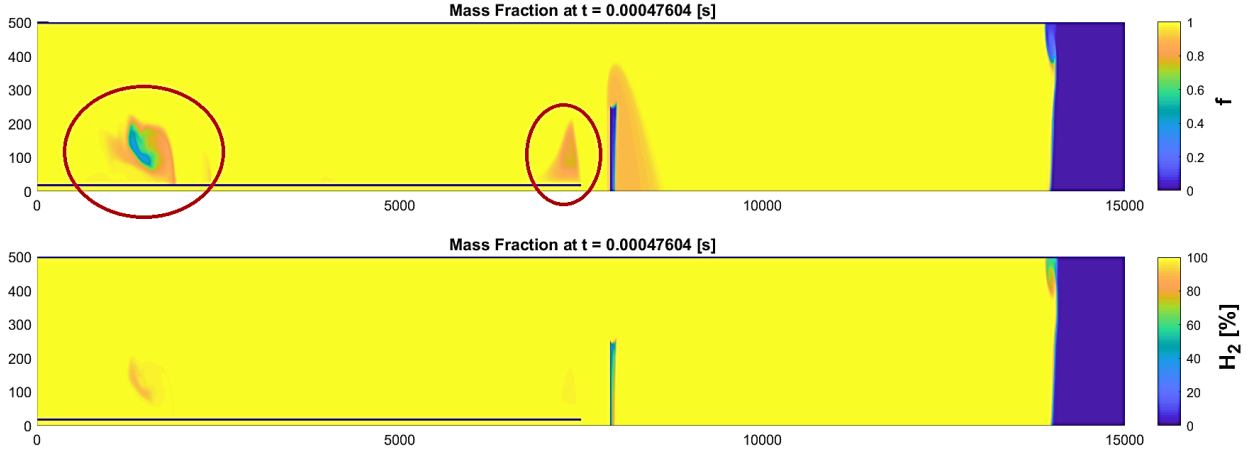


Figure 3.9: Potentially hazardous regions in a developed flow in the channel at $t = 4.7604e - 04\ [s]$

Despite all the discussions in this section, as mentioned in the beginning the results of the USN-FLIC code simulation seem to be overestimated. In order to make sure that the simulated temperature contours are showing values noticeably higher than the real case, validation with the shock-tube problem has been carried out for this method that its results will be exceptionally given in the next chapter because first, explanations about the utilized rigorous thermodynamic

models and the Helmholtz function should be given, the shock-tube problem can be discussed. Along with this validation, for the sake of having more data to compare and evaluating the results of the discussed method, it is tried to simulate the same scenario by using an open-source C++ toolbox, called Open-source Field Operation and Manipulation (OpenFOAM) which is a CFD software. The results and discussions about this attempt will be presented in the next chapter.

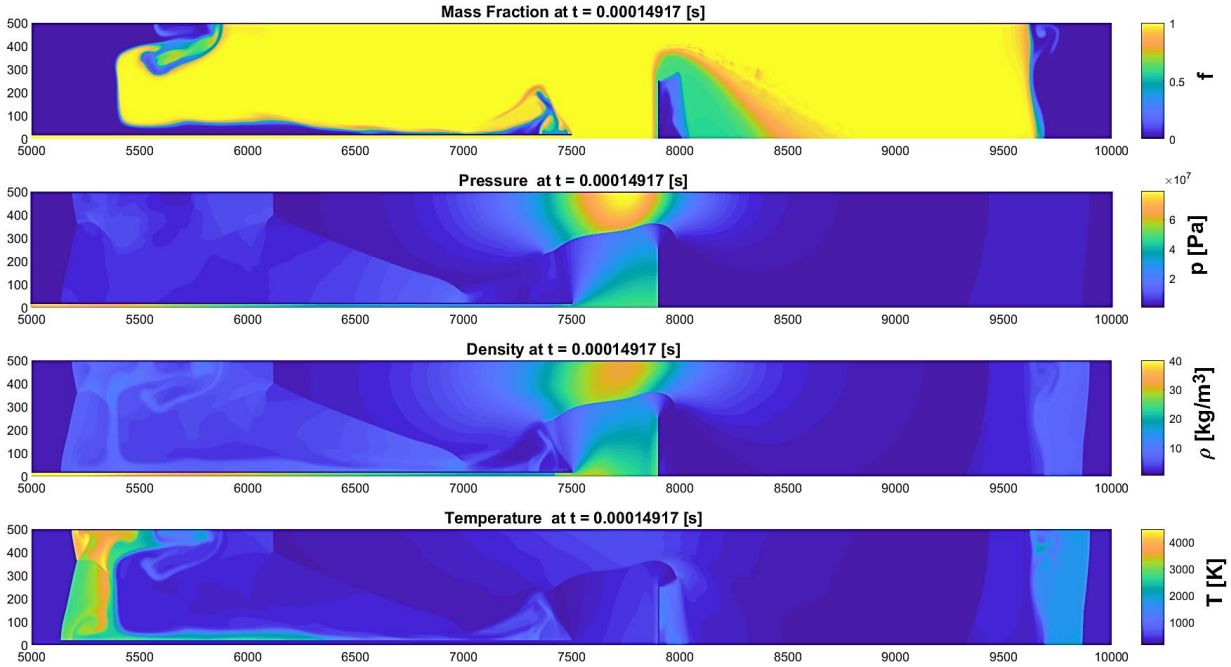


Figure 3.10: Flow properties of the simulation at $t = 1.4917e - 03\ [s]$, as a comparison to *Figure 3.6*

4 OpenFOAM Simulations

This chapter is about what was done to enable OpenFOAM to simulate high-pressure hydrogen leakage into the air. The first half of this chapter is about the OpenFOAM solvers and thermophysical models including their constraints and governing equations, in addition to the reasons behind choosing the utilized solver. The second half is about the simulation results and their validation.

4.1 Solver Selection

There are several criteria for choosing a proper solver to simulate the desired scenario in OpenFOAM which is tried to categorize in the following subsections. For understanding the below discussion, in the beginning, it is important to note what equations exactly are going to be solved by the solver and these equations contain what mathematical terms. Generally, the equations solved in all CFD simulations are so-called the Navier–Stokes equations. This set of equations are in fact in the form of conservation equations discussed in *Chapter 2*. They are mainly 4 transport equations for mass, velocity, and energy. But in the case of having a compressible flow (variable density), an extra equation for relating the density to the temperature and pressure must be considered. The general form of these transport equations for a specific property of the flow (ϕ) can be written as *Equation (4.1)*. In this equation Γ_ϕ is the diffusion coefficient of ϕ which will be viscosity (μ) in terms of momentum conservation equations and thermal conductivity (k) in terms of energy conservation equation. In addition to these 5 equations (4 transport equations + 1 equation of state), in order to deal with the flow fluctuations (turbulence), the so-called Reynolds decomposition equations will also be considered which will be more explained in the turbulence model section [25].

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\substack{Rate\ of\ change\ of\ \phi \\ in\ fluid\ element \\ (\textbf{\textit{Transient term}})}} + \underbrace{\vec{\mathbb{u}} \cdot \nabla(\rho\phi)}_{\substack{Net\ rate\ of\ flow\ of\ \phi \\ out\ of\ fluid\ element \\ (\textbf{\textit{Convection term}})}} = \underbrace{\nabla \cdot (\Gamma_\phi \nabla\phi)}_{\substack{Rate\ of\ change\ of\ \phi \\ due\ to\ diffusion \\ (\textbf{\textit{diffusion term}})}} + \underbrace{S_\phi}_{\substack{Rate\ of\ change\ of\ \phi \\ due\ to\ sources \\ (\textbf{\textit{Source term}})}} \qquad (4.1)$$

Each solver in a CFD code makes its specific assumptions and utilizes special numerical schemes to manage each term of these equations. But this procedure is a compromise between the importance of these terms. Therefore, concerning having a stable and accurate solution in a specific case, according to the flow conditions a particular kind of solver should be chosen to emphasize particular aspects of the flow.

4.1.1 Large gradients and Mass Convection

First and foremost, as discussed in the previous chapters, the dominant phenomenon in this simulation is shock waves. Therefore, not only there will be sharp discontinuities in the flow, but also in a comparison between the convection and diffusion terms of conservation equations, the former one prevails which can be proved by calculating the Peclet number. Regarding the shock waves, it is known that at least there are regions in the hydrogen flow in which the flow

velocity is equal or greater than the speed of sound. So, as a rough estimate to compute the minimum Peclet number of the flow based on *Equation (4.2)*, the velocity (u) can be set as the speed of sound in hydrogen (at $1\,atm$, and $300\,K$) which is around $1319\,m/s$, the characteristic length (L) can be set as the grid resolution which is considered similar to the previous chapter $1e-04\,m$, and finally the mass diffusion coefficient (D) should be set to $6.1e-05\,m^2/s$ for hydrogen [26], [27]. Based on these values the Peclet number of the flow will be at least equal to 2162.3 which is much greater than zero.

$$Pe = \frac{L.u}{D} \tag{4.2}$$

As a result of all these, the desired solver in addition to sharp discontinuities should also be able to handle the dominant convection term. In other words, in the best case for dealing with both issues, this solver should contain a Riemann scheme to solve the convective fluxes.

4.1.2 Compressibility

Hydrogen at low pressures like most gases behaves as an ideal gas, however, in this project, the pressure of the hydrogen reservoir is $70\,MPa$. Thus, it is important to check whether the effect of compressibility is significant or not. For this purpose, usually, a parameter called compressibility factor (Z) is defined based on the following equation for gasses [28].

$$Z = \frac{p \cdot V}{n \cdot R \cdot T} = \frac{p}{\rho \cdot \left(\frac{R}{MW}\right) \cdot T} \tag{4.3}$$

In this equation, pressure ($p\,[Pa]$), temperature ($T\,[K]$), and density ($\rho\,[kg/m^3]$) are properties of the real gas, $R = 8.314\,[kJ/(kmol.K)]$ is the universal gas constant, and $MW\,[kg/kmol]$ is the molecular weight of the gas. Considering this equation, the hydrogen compressibility factor in various pressures and temperatures is shown in *Figure 4.1*. As the pressure goes up to $70\,MPa$, the compressibility factor at $300\,K$ tends to 1.46.

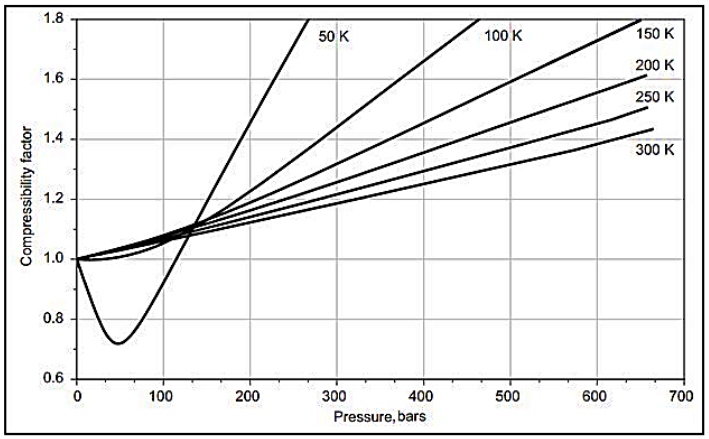


Figure 4.1: Hydrogen compressibility factor in various pressures and temperatures [28]

In order to make it clear how much error will be produced by using the ideal-gas EOS[1] and neglecting the effect of the compressibility factor, a comparison has been made between the

[1] Equation of state

calculated density by the ideal-gas law and the real (actual) density of hydrogen. As illustrated in *Figure 4.2* by yellow dots, when the pressure goes up to 70 MPa (at 300 K), as it is expected based on the discussed compressibility factor, density will be calculated with almost 50% error by the ideal-gas law while a more accurate EOS like Abel-Nobel can compute this value with only 1% error. For hydrogen, the Abel-Nobel EOS is a better model for pressures less than 200 MPa and temperatures higher than 150 K [29].

With respect to this discussion, the desired solver in addition to the previously mentioned features should also be able to consider compressibility.

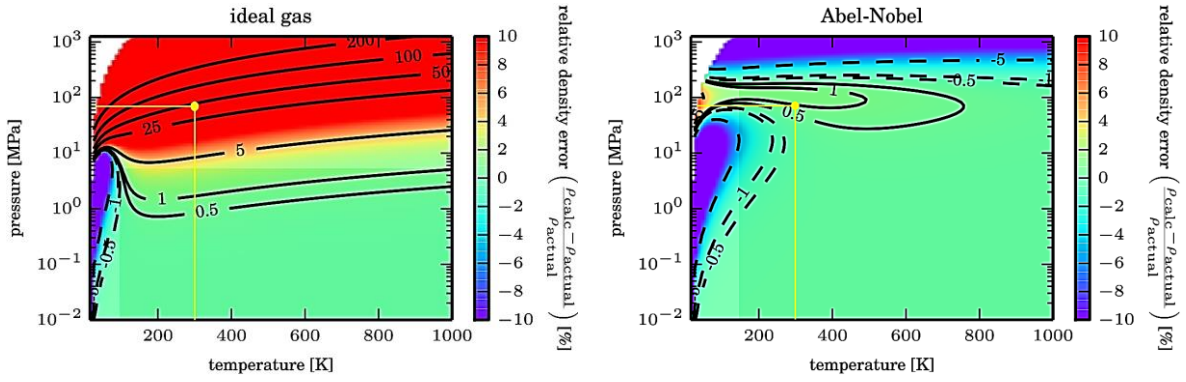


Figure 4.2: Relative density error in various pressures and temperatures for (a) ideal-gas EOS (b) Abel-Nobel EOS [29]

4.1.3 Multi-Component Mixtures

For simulating the intended scenario, the desired solver should be able to deal with multi-component mixtures in which the concentration of each component is variable. As the result, the only option of OpenFOAM for considering such cases is what is called *reactingMixture*. Since in other types of mixtures implemented in OF[1] like *pureMixture*, the concentrations of components are constant. Consequently, the desired solver has to solve the species transport equations, which enables the solver to also study the chemical kinetics in the simulation, if necessary.

Furthermore, it is necessary to add that regarding the wide temperature range of this problem (extremely low temperatures in the expanded-hydrogen regions and very high temperatures in the compressed-air regions) considering a multi-phase scenario in the simulation would be a privilege to have more accurate results. But due to the complexity of this problem, this topic is neglected in this project.

4.1.4 Final Decision

Based on all these discussions, the best available, built-in solvers in OpenFOAM v.7 for this case are *rhoCentralFoam* and *rhoPimpleFoam*. Between these two, the former one is much

[1] OpenFOAM

more suitable for this project. Because *rhoCentralFoam* exclusively deals with the convective fluxes based on the Kurganov and Tadmor schemes which were discussed in *Section 2.4.3*. But unfortunately, neither of them can handle the mentioned *reactingMixture*. So, in fact, neither of them can be utilized for simulating this scenario. The possible approaches to encounter this problem were to either write an original solver by the author or search and find a properly developed solver by others and make it suitable for this specific scenario. With respect to the existing constraints, the second approach was chosen.

4.2 Discovered Solvers and The Properties of the Chosen One

According to OF being an open-source CFD code, in general[1], there are several related developed solvers that can be found on the internet. A shortlist of them is given below:

- rhoReactingCentralFoam [30]
- hybridCentralSolvers [31]
- pisoCentralFoam [32]
- AeroFoam [33]
- hyStrath [34]

There were also other solvers, but due to lack of proper citation, they have not been listed here. Among these solvers, considering being matched with OF v.7 [2] as well as being suitable for simulating most aspects of the desired scenario, the *rhoReactingCentralFoam* was chosen. This solver was initially implemented to capture hydrogen detonation and basically, it is a straightforward combination of *rhoCentralFoam* and *reactingFoam* solvers to gain benefits from both of them, the central upwind scheme from *rhoCentralFoam*, and the chemical kinetics from *reactingFoam*. Although this solver was the most recent one in the above shortlist and had the least problems during compilation and also needed the least adjustments to become generally appropriate, it has almost all the features that are needed to simulate the intended case except alowing to use a non-ideal equation of state. So, to cope with this problem, a new thermophysical model had to be implemented in OF which will be discussed later in *Section 4*. In the following, the main procedure carried out by this solver will be described.

4.2.1 rhoReactingCentralFoam Governing Equations

In *rhoReactingCentralFoam* solver, in each iteration after calculating the surface properties based on the primitive fields data, the following steps will be taken. It should be mentioned that in the following equations the subscripts i, j, and k are showing a vector component along the i, j, and k directions. However, the I, and V subscripts are abbreviations for inviscid and viscous [35].

1. Solving mass conservation equations to provide a new value for density (ρ). This equation will be solved based on the velocity of the previous time step.

$$\frac{\partial \rho}{\partial t} + \frac{\partial (u_i \rho)}{\partial t} = 0 \tag{4.4}$$

[1] Without considering the version of the utilized OpenFOAM.

[2] Having less trouble in compilation compare to the other solvers.

2. Solving momentum conservation equations to provide new values for the velocities (u_i). This step will be done in two sections:
   2.1. Solving the inviscid momentum equation and updating the velocities values (u_i) based on the updated density value from the previous step.

$$\left(\frac{\partial(\rho u_i)}{\partial t}\right)_I + \frac{\partial(\rho u_i u_j)}{\partial x_j} + \frac{\partial p}{\partial x_i} = 0 \tag{4.5}$$

   2.2. In the case of having a viscous flow, solving the diffusion correction equation based on the calculated velocities in the previous inviscid section (u_i^I) to find new values.

$$\left(\frac{\partial(\rho u_i)}{\partial t}\right)_V - \frac{\partial}{\partial x_j}\left(\mu\frac{\partial u_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\mu\left(\frac{\partial u_j^I}{\partial x_i} - \frac{2}{3}\frac{\partial u_k^I}{\partial x_k}\delta_{ij}\right) = 0 \tag{4.6}$$

3. Solving species transport equations to provide new values for the concentrations of each of the components (Y_i). This step, however, will be done in one of the following ways:
   3.1. In the case of having an inviscid flow, solving the inviscid species equations based on the updated density and velocities and renew the concentrations. $R_{Y_i}$is the reaction rate of the species i.

$$\frac{\partial(\rho Y_i)}{\partial t} + \frac{\partial(\rho u_j Y_j)}{\partial x_j} - R_{Y_i} = 0 \tag{4.7}$$

   3.2. In the case of having a viscous flow, the above transport equation will have an extra diffusion correction term to calculate new concentrations.

$$\frac{\partial(\rho Y_i)}{\partial t} + \frac{\partial(\rho u_j Y_j)}{\partial x_j} - R_{Y_i} - \frac{\partial}{\partial x_j}\left(\mu\frac{\partial Y_i}{\partial x_j}\right) = 0 \tag{4.8}$$

4. Finally, solving the energy conservation equation to find new values for temperature and pressure. This step, like *step 2*, will be done in two sections:
   4.1. Solving the inviscid energy equation based on the updated density and velocities to find a new value for the total energy (E).

$$\left(\frac{\partial(\rho E)}{\partial t}\right)_I - \frac{\partial}{\partial x_k}\big(u_k(\rho E + p)\big) - \frac{\partial}{\partial x_i}\mu u_j\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) = 0 \tag{4.9}$$

   Then calculating a new temperature based on the following equation and updated total energy.

$$T = \frac{1}{C_v}\left(E - \frac{u_k^2}{2}\right) \tag{4.10}$$

   4.2. In the case of having a viscous flow, then a diffusion correction equation will be solved for the temperature to correct the value.

$$\left(\frac{\partial(\rho C_v T)}{\partial t}\right)_V - \frac{\partial}{\partial x_k}\left(k\frac{\partial T}{\partial x_k}\right) = 0 \tag{4.11}$$

   At last, the pressure will be calculated based on the EOS and the updated temperature. The diffusion coefficients (μ and k) also will be updated based on this temperature.

4.3 Implementation of a New Thermophysical Model

As it was mentioned in the previous section, for the chosen solver there was no thermophysical model to deal with non-ideal equations of state. Therefore, it was necessary to implement a suitable one based on the requirements of this project. Thermophysical models in OF are packages of numerical schemes that deal with the energy, heat, and physical properties of the case. In each package the following thermophysical properties should be determined [36]:

- Selecting the class of the thermophysical model which controls the type of the mixture, reaction, and compressibility (set under the keyword *type*)
- Selecting the mixture model which specifies the mixture composition (set under the keyword *mixture*)
- Selecting the transport model which evaluates the diffusion coefficients, i.e. dynamic viscosity, thermal conductivity, and thermal diffusivity (set under the keyword *transport*)
- Selecting the thermodynamic model to determine how to evaluate the specific heat capacity of the constituents (set under the keyword *thermo*)
- Specifying the composition of each constituent (set under the keyword *specie*)
- Selecting the equation of state (set under the keyword *equationOfState*)
- Selecting the form of energy that is going to be used in the solution, either internal energy or enthalpy (set under the keyword *energy*)

Among the built-in equations of state, the most proper one for coping with the compressibility of this simulation is the Peng-Robinson EOS. Thus, a new thermophysical model including the Peng-Robinson EOS was implemented by adding required entries in the corresponding paths and recompiling the *basic*, *reactionThermo*, and *specie* libraries of the *thermophysicalModels* library. The mentioned steps are concisely described below [37]:

1. Path:
   `$(WM_PROJECT_USER_DIR)/src/reactionThermo/psiReactionThermo/psiReactionThermos.C`
   Entry:
   ```
   makeReactionThermos
   (
       psiThermo,
       psiReactionThermo,
       hePsiThermo,
       reactingMixture,
       sutherlandTransport,
       sensibleInternalEnergy,
       janafThermo,
       PengRobinsonGas,
       specie
   );
   ```
2. Path:
   `$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/include/thermoPhysicsTypes.H`
   Entry:
   ```
   typedef
   ```

```
sutherlandTransport
<
   species::thermo
   <
        janafThermo
        <
             PengRobinsonGas<specie>
        >,
        sensibleInternalEnergy
   >
> nonIdealGasEThermoPhysics;
```

3. Path:
   `$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/include/reactionTypes.H`
   Entry:
   `typedef Reaction<nonIdealGasEThermoPhysics> nonIdealGasEReaction;`
4. Path:
   `$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/reaction/reactions/makeReactions.C`
   Entry:
   `makeReactions(nonIdealGasEThermoPhysics, nonIdealGasEReaction)`
5. Path:
   `$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/reactionThermo/chemistryReaders/chemistryReader/makeChemistryReaders.C`
   Entry:
   `makeChemistryReader(nonIdealGasEThermoPhysics);`
   `makeChemistryReaderType(foamChemistryReader, nonIdealGasEThermoPhysics);`

By implementing this thermophysical model, now everything is ready to step forward and simulate the desired scenario by the chosen solver.

4.4 Initial Simulations Setups

In the initial steps, two main simulations were done with OF to compare the initial hydrogen flow rate of a two-dimensional simulation with a pseudo-three-dimensional one. The boundary and initial conditions, and all the setups were exactly the same between these two, but the differences were that the first simulation was an axisymmetric 2D-simulation, however, the second one was axisymmetric by using a wedge geometry, and also in the pseudo-3D-simulation for simplicity (without giving away the main goal) the obstacle was eliminated from the domain.

4.4.1 Geometry and Boundary Conditions

4.4.1.1 Two-Dimensional

The general shape of the simulation geometry was again chosen based on the available experimental apparatus in the Combustion Laboratory. But this time, the axisymmetric option was available. So, the geometry was created by the *blockMesh* tool of OF in $x - y$ plane with considering the entire bottom boundary as *symmetryPlane*. Also, it is worth mentioning that the dimension of the geometry in $z - direction$ was chosen based on the fact that the surface of the nozzle, which is a rectangular surface in this case, is equal to the 3D circular one. Thus, the $z - direction$ dimension was set to $2.3562\ mm$. All the other dimensions and boundary conditions in $x - y$ plane are the same as the previous simulation, shown in *Figure 3.1*.

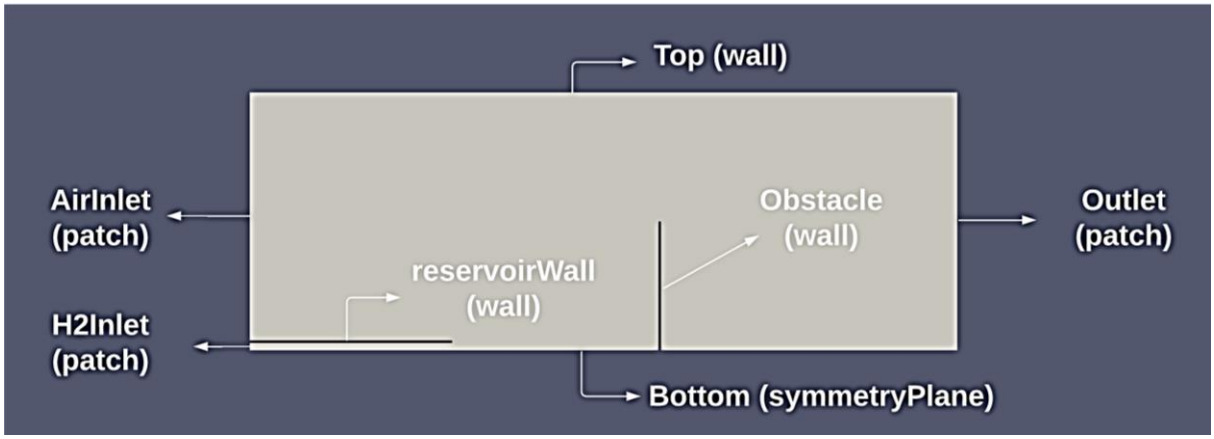


Figure 4.3: Boundary names conditions of the initial 2D-simulation
(Dimensions are not real)

4.4.1.2 Pseudo-Three-Dimensional

As mentioned above the differences between the geometry of 2D- and pseudo-3D-simulations were, firstly, in the pseudo-three-dimensional wedge geometry the front and back boundary conditions were set as *wedge* patches instead of *empty* patches set in 2D-simulation. And secondly, the obstacle was eliminated from the domain to have a simpler geometry. For more illustration, a schematic of such a geometry and boundary conditions is shown in *Figure 4.4*.

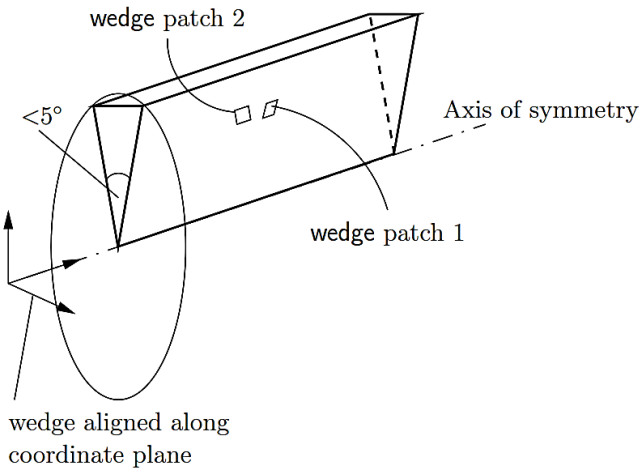


Figure 4.4: Boundary conditions of a two-dimensional wegde geometry [36]

As it is pointed out in *Figure 4.4*, the small angle between wedge planes must be less than 5° to have an accurate solution. In this simulation, the angle was set to be 1° which due to the Courant numbers makes the simulation too much time-consuming.

The detailed boundary conditions of the main flow properties are tabulated below.

Table 4.1: Details of the boundary conditions

Boundary Names	Boundary types	Boundary Conditions of the Flow Properties			
		U $[m/s]$	p $[Pa]$	T $[K]$	H2, O2, N2, Ydefault [mass fraction]
Top	wall	noSlip	zeroGradient	zeroGradient	zeroGradient
Bottom	symmetry Plane (/ empty)	symmetry Plane (/ empty)	symmetry Plane (/ empty)	symmetry Plane (/ empty)	symmetry Plane (/ empty)
H2Inlet	patch	pressureDirectedInletVelocity	fixedValue	fixedValue	fixedValue
AirInlet	patch	inletOutlet	totalPressure	inletOutlet	inletOutlet
Outlet	patch	inletOutlet	totalPressure	inletOutlet	inletOutlet
reservoirWall	wall	noSlip	zeroGradient	zeroGradient	zeroGradient
Obstacle	wall	noSlip	zeroGradient	zeroGradient	zeroGradient
frontAndBack	empty (/ wedge)	empty (/ wedge)	empty (/ wedge)	empty (/ wedge)	empty (/ wedge)

4.4.2 Mesh

The grid resolution of these simulations at the center of the domain[1] was uniformly set, based on the mesh independence test of the USN-FLIC code simulation, to 100 μm. Although, by using the grading meshes the cell sizes at $AirInlet$ and $Outlet$ boundaries reached 200 μm. The type of the entire mesh is a structured quadrilateral mesh or more precisely structured hexahedron; because OF even considers 2D-simulations as 3D with one cell in the third direction (in this case $z - direction$), so there is no actual 2D-mesh.

Due to the very high resolution of the mesh and wide range of dimensions, it is not practical to show the generated mesh here. However, the total number of cells in these cases is 4,819,750.

[1] Between the outlet of the reservoir and the obstacle.

4.4.3 Initial Conditions

The most important part of this section is the usage of the *setFields* dictionary. If the pressure of the hydrogen outlet had been set to a fixed value equal to $70\ MPa$, the solver would have encountered an extremely large pressure gradient at this point, from $70\ MPa$ to $0.1\ Mpa$ (ambient pressure). Which may lead to solver divergence due to the fixed value of the outlet. Therefore, as it was also mentioned in *Chapter 3*, in order to prevent this scenario, a long entry was considered for the hydrogen in which the initial conditions were set to be equal to the reservoir conditions. But in this case, since the domain properties of this part are not fixed, a backward propagation of the pressure wave can form to the hydrogen inlet which allows the large pressure gradient be smoothly damped.

Setting the initial conditions of a specific region in the computational domain can be done by utilizing the *setFields* dictionary in OF. Base on this, as can be observed in *Figure 4.5* for the pressure, the initial conditions of the entire domain except for the hydrogen entry were set as $p = 1e + 05\ [Pa], T = 291\ [K], U = 0\ [m/s], H_2 = 0, O_2 = 0.21$, and $N_2 = 0.79$. However, for the hydrogen entry, the initial conditions were set to $p = 7e + 07\ [Pa], T = 291\ [K]\ , U = 0\ [m/s], H_2 = 1, O_2 = 0$, and $N_2 = 0$.

It is strange to mention that in the case of using *ASCII* format for writing the data in this simulation, the *setFields* dictionary will not perform properly at the explained initial conditions will not be generated. The reason behind this may be a large number of cells in the generated mesh. To avoid this problem the *Binary* format should be utilized.

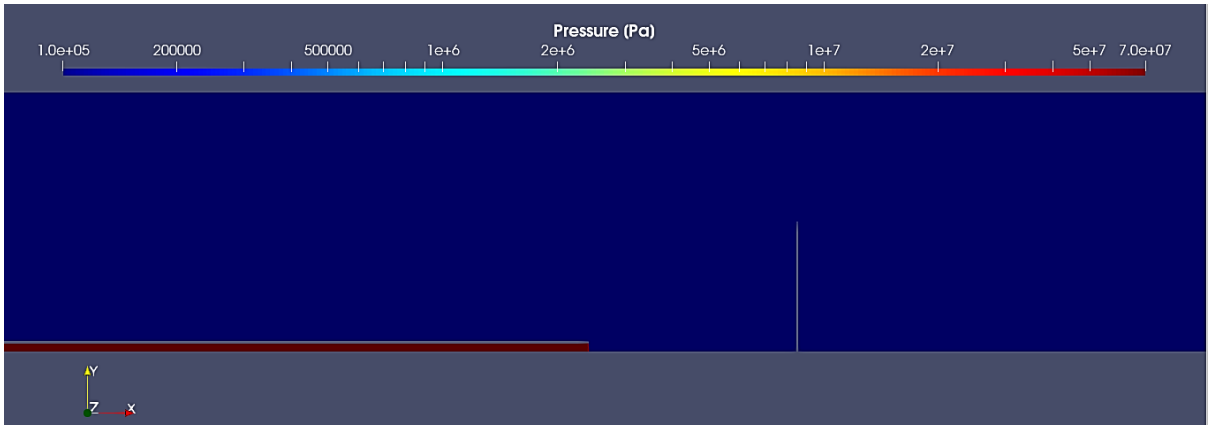


Figure 4.5: Pressure initial condition at the center of the domain

4.4.4 System Sub-Dictionaries

The settings of the solver and numerical schemes are generally determined in the *system* library. From case to case the including files in this library will be different, but for these simulations, the following setups have been adjusted. The details of the sub-dictionaries are given in *Appendix E*.

4.4.4.1 controlDict

The so-called global parameters of the main solver are configured in the *controlDict* dictionary. As well as specifying some routines in this file, like the time step which is set to be adjustable, what in this case is special is the usage of acoustic Courant number in addition to the discussed

CFL for adjusting the time step. And by doing this, the time step will be controlled by the speed of shock waves. The values of the maximum acoustic Courant number (*maxAcousticCo*) and the maximum central Courant number (*maxCo*) are set based on the other studies including the Courant number tests done by McGough [38].

Furthermore, the post-processing functions which will be used to analyze the results should be implemented in this file. The one that is used in this project is the *minMaxMagnitude* function to find the value and the location of the maximum temperature in each iteration.

4.4.5 fvSchemes

The numerical schemes used for each term of the transport equations are set in the *fvScheme* dictionary. As discussed before, the characteristic of all the solvers in the *rhoCentralFoam* family is that they use central upwind schemes (Kurganov and Tadmor) to deal with the convective fluxes. In addition to this, what in this simulation is special is the usage of the *vanLeer* scheme for divergence schemes (except for the velocity and τ_{MC}) and interpolation schemes. The *vanLeer* scheme is a TVD flux limiting scheme like other schemes discussed in *Section 2.4*, but it uses different criteria for limiting the convective fluxes at the interfaces.

In the end, it is good to mention that the divergence schemes are numerical schemes concerned about the advection term of the transport equations while the interpolation schemes are the numerical schemes used to interpolate the flow properties' values at the interfaces by using the averaged values at the cell centers.

4.4.6 fvSolution

The solvers utilized for each of the flow properties, as well as the tolerances in solving each equation and the parameters of the main solver algorithm (like SIMPLE, PIMPLE, etc.) are determined in the *fvSolution* dictionary. The special case in this section is the usage of the Preconditioned Bi-conjugate Gradient Stabilized (*PBiCGStab*) solver for velocity and species equations. The reason behind this is to accelerate and smooth the convergence procedure.

Besides, due to the very large pressure gradients at the beginning of the simulation, the *relaxationFactors* has to be used to avoid the *Floating point exception* error. This sub-dictionary by controlling the value of under-relaxation factors for the equation of each flow property will help to stabilize the solution.

4.4.6.1 decomposeParDict

At last, for accelerating the simulation and using all power of the computer system, the *decomposePar* dictionary was used to divide the computational domain into several sections and solve the transport equations in each section individually by a specific core of the CPU. In these simulations, the domain was divided by the *simple* method into 16 sections to simulate the case in the parallel mode.

4.4.7 Constant Sub-Dictionaries

The *constant* dictionary in general contains all the physical and chemical information that is needed to solve the transport equations. Its sub-dictionaries would be different from case to case. But in the original *rhoReactingCentralFoam* tutorial, the subdictionaries are as follows:

4.4.7.1 chemistryProperties

In the case of considering chemical reactions in the simulation, the type of the reactions as well as the utilized solver for them and tolerances will be determined in the *chemistryProperties* dictionary. But in the initial simulations of this project, in order to simplify the simulations and achieving faster results, this option was deactivated by setting the *chemistry* entry to *off*.

4.4.7.2 combustionProperties

As the name implies, in the case of having combustion in the simulation the combustion model and its corresponding coefficients will be set in the *combustionProperties* dictionary. But as explained above any chemical reaction in the initial simulations of this project was neglected.

4.4.7.3 reactions

In the *reactions* dictionary, every species included in the simulation as well as their composing elements should be defined. Besides, the chemical kinetics of every reaction which is to be considered in simulation should be defined in detail (specifying the pre-exponential factor, activation temperature, etc.) under the keyword entry *reactions*.

Although, as discussed above, the chemical reactions are neglected in this step, to be able to have multi-component mixtures with variable concentrations of the components, the elements and species should be determined in this dictionary. In this stage of the project, the considered species are H_2, O_2, and N_2 and consequently, the composing elements will be H, O, and N.

4.4.7.4 thermophysicalProperties and thermo.compressibleGas

The *thermophysicalProperties* dictionary mainly contains the thermophysical model of the simulation which was discussed in *Section 4.3*. Also, it specifies the *inertSpecie*, the *chemistryReader*, *foamChemistryFile*, and *foamChemistryThermoFile*. The *foamChemistryFile* is pointed to the location of the *reactions* dictionary and the *foamChemistryThermoFile* is pointed to the location of the *thermo.compressibleGas* dictionary. The latter dictionary contains the coefficients that are needed for the utilized thermophysical model as we as the chemical properties of components (elements and species).

The thermophysical model used in these two initial simulations is the one that was implemented in *Section 4.3*. the details of this model are as follows:

- *type*: The type of the class of this model was set to *hePsiThermo* which is for reacting mixtures based on the compressibility ($\psi = (RT)^{-1}$).
- *mixture*: The mixture type was set to *reactingMixture*; as it was discussed, this feature of the utilized solver was the main reason behind this choice.
- *transport*: The transport model was selected as *sutherland* in which the dynamic viscosity of each component for temperature T is calculated based on the following equation [36]:

$$\mu = \frac{A_s\sqrt{T}}{1 + T_s/T} \tag{4.12}$$

In this equation, A_s and T_s are the so-called Sutherland coefficients that should be determined for each component in the *thermo.compressibleGas* dictionary. After this, the thermal conductivity will be calculated based on this dynamic viscosity (μ) and the specific heat capacity at constant volume (C_v) as follows:

$$k = \mu \cdot C_v(p,T)\left(1.32 + \frac{1.77R}{C_v(p,T)}\right) \tag{4.13}$$

If necessary, the thermal diffusivity (α) will also be computed based on this calculated thermal conductivity (k) and the specific heat capacity at constant pressure (C_p).

- *thermo*: the thermodynamic model was selected to be *janaf* in which the specific heat capacity at constant pressure (C_p) for temperature T is calculated based on the following polynomial which is for the ideal-gas part of C_p (the real gas part will be added to this value based on the chosen equation of state) [36]:

$$C_p = R \cdot \Big(\big((a_4T + a_3)T + a_2\big)T + a_1\Big)T + a_0\Big) \tag{4.14}$$

In this equation, the a_i coefficients are the so-called janaf coefficients that should be defined in the *thermo.compressibleGas* dictionary along with two more coefficients a_5 and a_6 which will be used to calculate the enthalpy and entropy based on the following equations (same as above these equations calculate the ideal-gas part of each value and the real-gas part will be added to them based on the utilized equation of state):

$$\frac{H}{RT} = \left(\left(\left(\left(\frac{a_4}{5}T + \frac{a_3}{4}\right)T + \frac{a_2}{3}\right)T + \frac{a_1}{2}\right)T + a_0\right)T + a_5 \tag{4.15}$$

$$\frac{S}{R} = \left(\left(\left(\frac{a_4}{4}T + \frac{a_3}{3}\right)T + \frac{a_2}{2}\right)T + a_1\right)T + a_0 \ln T + a_6 \tag{4.16}$$

There are two sets of janaf coefficients for each component, and each set corresponds to a specific range of temperature. Thus, it is needed to specify the limits of these intervals as T_{low}, T_{common}, and T_{high}.

- *equationOfState*: The EOS of this model was set to be *PengRobinsonGas* which calculates the density generally based on the following relation:

$$\rho = \frac{1}{z \cdot R \cdot T}p \tag{4.17}$$

In which the compressibility factor (z) is considered. In order to enable the Peng-Robinson EOS to calculate the real-gas part of thermophysical properties (e.g. as mentioned above for heat capacity, enthalpy, and entropy), some coefficients should be determined for each component in the *thermo.compressibleGas* dictionary. These coefficients are critical temperature T $[K]$, critical molar volume V_c $[m^3/kmol]$, critical compressibility factor Z_c, critical pressure p_c $[Pa]$, and acentric factor ω which are determined based on [39], for this project.

- *energy*: The energy equation solution was set as *sensibleInternalEnergy* and the reason was that based on the other studies this option will produce a less numerical error in comparison to the sensible enthalpy option.

The utilized values for the initial simulations as the Sutherland and Janaf coefficients as well as the critical properties needed for the Peng-Robinson equation of state are given in *Appendix F*.

4.4.7.5 turbulenceProperties

As mentioned in *Section 4.1*, for considering turbulence in the flow some extra equations need to be derived and used in order to solve the fluctuation terms. All in all, there are two main approaches implemented in OF, the Reynolds-Averaged Simulation (RAS) and the Large Eddy Simulation (LES) modelings. Each of them uses a different set of equations to capture the turbulence.

In the *turbulenceProperties* dictionary, this approach should be chosen. The model that is used in this project is called *kOmegaSST*. Which is a turbulence model for dealing with high Reynolds number flows. This turbulence model is actually a combination of the standard $k-\varepsilon$ and $k-\omega$ models in a way that gains the advantages of both models. Technically, it uses the $k-\omega$ model near the wall boundaries to capture the viscous sub-layer more accurately, while uses the $k-\varepsilon$ model away from these regions, i.e. in the far-field fully turbulent regions. The main reason behind this choice was that the $k-\omega\ SST$ model is more suitable than the others to deal with cases in which adverse pressure gradients, separations, swirls, and curvatures exist inside the flow field as what happens in this project due to the existence of the obstacle in the domain. The initial values of the k and ω parameters were calculated based on the following equations. There are rough estimates to initiates solving the turbulent transport equations [25], [36]. In these equations, the values of the flow properties (U, Re) were put according to the USN-FLIC code simulation results.

$$\begin{cases} k = \frac{3}{2}(UI)^2 \\ I = 0.16Re^{-1/8} \end{cases} \tag{4.18}$$

$$\begin{cases} \omega = C_\mu^{-1/4}\ \frac{\sqrt{k}}{l} \\ C_\mu = 0.09 \end{cases} \tag{4.19}$$

The boundary and initial conditions of the turbulence parameters, the turbulent kinetic energy (k), and the specific dissipation rate (ω), are tabulated on the next page in *Table 4.2*.

Now that all the properties and settings of the mentioned simulations, 2D and pseudo-3D, have been discussed and the reasons behind all the decisions have been reviewed, the next step which is the results of these initial simulations can be taken into account.

Table 4.2: Details of the boundary and initial conditions of the turbulence parameters

Boundary Names	Boundary types	Boundary Conditions of the Flow Properties	
		Turbulent Kinetic Energy (k $[m^2/s^2]$)	**Specific Dissipation Rate (ω $[s^{-1}]$)**
Top	wall	kqRWallFunction (uniform 0.01)	omegaWallFunction (uniform 0.1)
Bottom	symmetryPlane (/ empty)	symmetryPlane (/ empty)	symmetryPlane (/ empty)
H2Inlet	patch	zeroGradient	zeroGradient
AirInlet	patch	zeroGradient	zeroGradient
Outlet	patch	zeroGradient	zeroGradient
reservoirWall	wall	kqRWallFunction (uniform 254.217)	omegaWallFunction (uniform 9.703e03)
Obstacle	wall	kqRWallFunction (uniform 0.01)	omegaWallFunction (uniform 0.1)
frontAndBack	empty (/ wedge)	empty (/ wedge)	empty (/ wedge)

4.5 Initial Simulations Results and Discussion

In order to demonstrate the general structure of the results of these simulations, the flow properties of the 2D-simulation are shown in *Figure 4.6*. It is important to note that for having a better visualization of the changes of flow properties, except for the mass fraction diagram, other contours are shown in a logarithmic scale. As can be seen in *Figure 4.6*, the hydrogen-air turbulent mixing is simulated far better than the USN-FLIC code simulation, and the wrinkling edges of the hydrogen cloud can be obviously observed in the hydrogen mass fraction diagram. Another significant aspect of these simulations is that the flow properties in the near-wall regions are calculated more accurately than the previous simulation results, and the reason is that in spite of the USN-FLIC code, the OF solver can handle the calculations of the transport equations in boundary layers by considering the viscous sub-layer[1]. As a result of this feature, it can be seen that in the near-wall regions not only there is a laminar thin layer of air compressed by the hydrogen cloud, but also there are spots where the air bubbles are trapped between the hydrogen cloud and walls. In general, this detailed information from the hydrogen-air mixing procedure shows more potentially combustible regions in the flow than the USN-FLIC code simulation results which will be discussed in the following.

According to the 2D-simulation, the highest temperature of this scenario ($2500\ K$) occurs at the beginning of the hydrogen release, and after the shock waves hit the *Obstacle* and *Top* boundaries, the temperature in the near-wall regions could not exceed this highest temperature.

[1] It was mentioned in the turbulent properties section that on the features of using the $k-\omega\ SST$ turbulence model is that it can manage the viscous sub-layer better than the other RAS models.

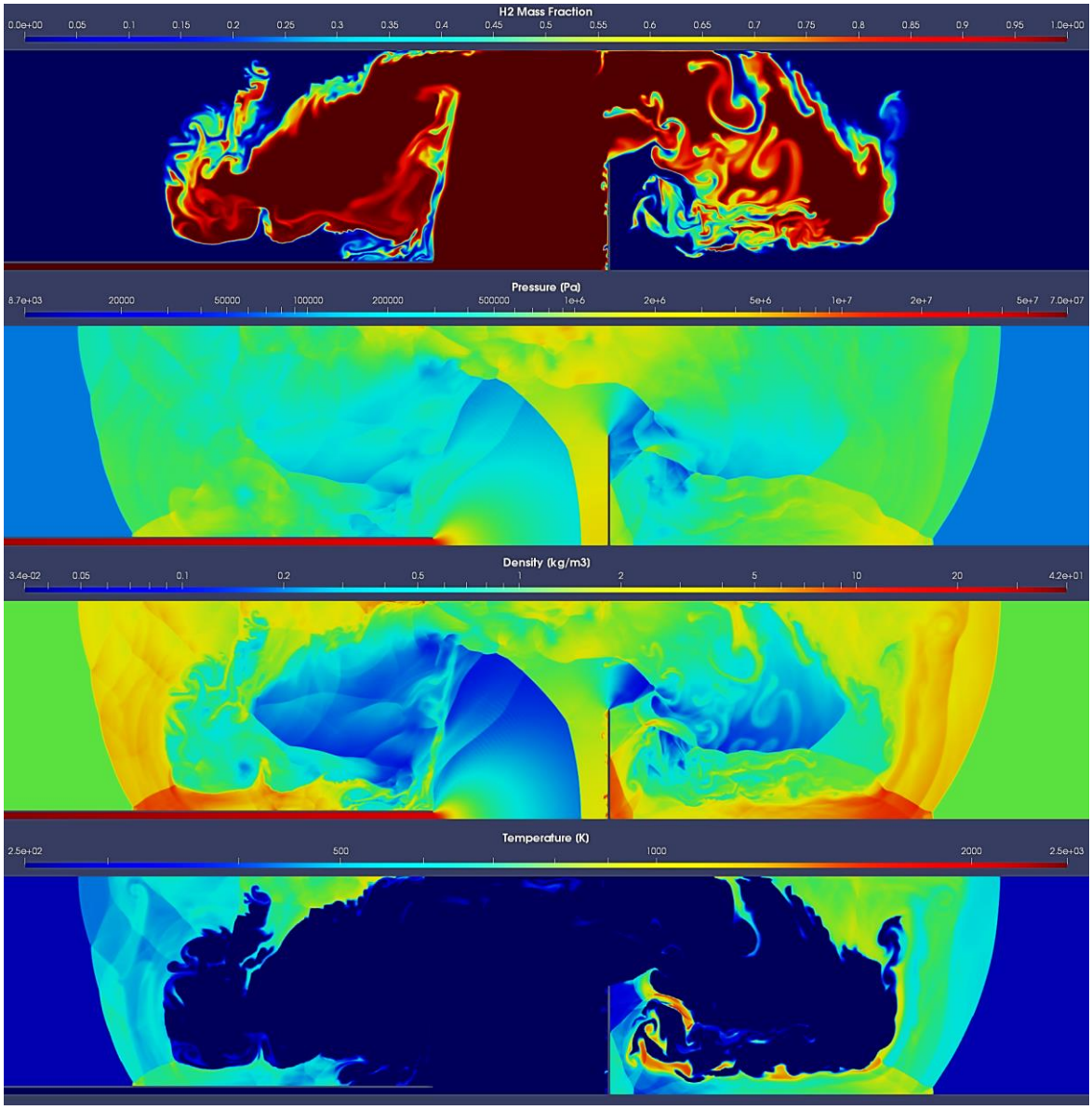


Figure 4.6: Flow properties of the OF initial 2D-simulation at $t = 1.26e-04\ [s]$

Another important facet of these simulations is that as the flow propagated through the channel, some regions were formed in which the pressure decreases far below the atmospheric pressure. For example, in the above diagrams, the minimum pressure of the flow field is equal to $8700\ Pa$ which is corresponding to $65.26\ mmHg$. And this matter is important according to the pressure effect on hydrogen self-ignition. As shown in *Figure A.1* (*Appendix A*), it can be seen that when the pressure goes below $300\ mmHg$, the critical temperature for hydrogen spontaneous ignition starts to reduce. So, any further pressure reduction in the flow field below this $65.26\ mmHg$, could make the hydrogen-air mixture highly combustible which should be discovered by simulating this case over a longer time.

Similar to what was done on the USN-FLIC code simulation results, if the areas with hydrogen concentration between $5 - 75\ \%$ are considered, as depicted in *Figure 4.7*, it can be observed that the temperature of these regions, especially behind the *Obstacle*, exceeds $1000\ K$ which could lead to a hazard because this region due to the existence of the obstacle has the best

mixing conditions in the flow field. Considering a cell even with a very low hydrogen concentration in this region ($0.05 < x_{H_2} < 0.75$ and $T > 1000\ [K]$), like the one with 7% hydrogen, $1152.34\ K$ temperature, and $1.325\ MPa$ pressure, shows that the spontaneous ignition of hydrogen can occur in less than $70\ \mu s$ in this area.

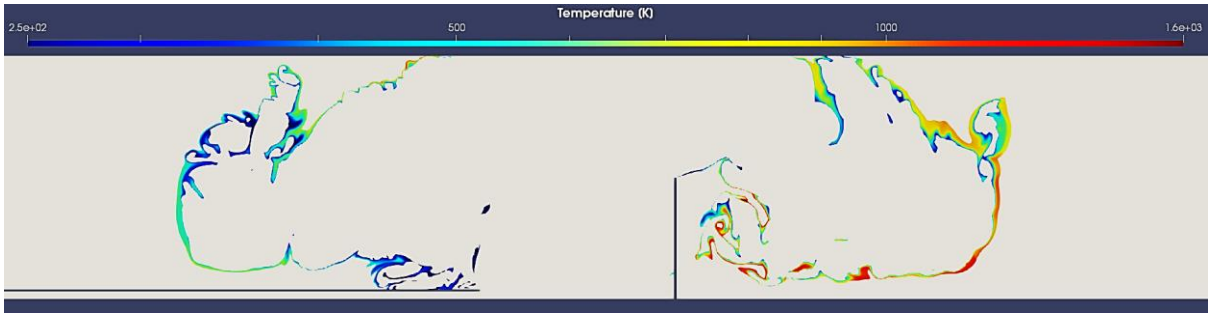


Figure 4.7: Boundary edges of the hydrogen cloud with the hydrogen concentration between $5 - 75$ %

By using the Cantera chemical kinetics toolbox, it can be revealed that the combustion of this particular hydrogen-air mixture will happen like the following. Also, it is good to mention that although the $68\ \mu s$ induction time is relatively long, according to the existence of large eddies in this area, this could be achieved given that this sample point has the least hydrogen concentration.

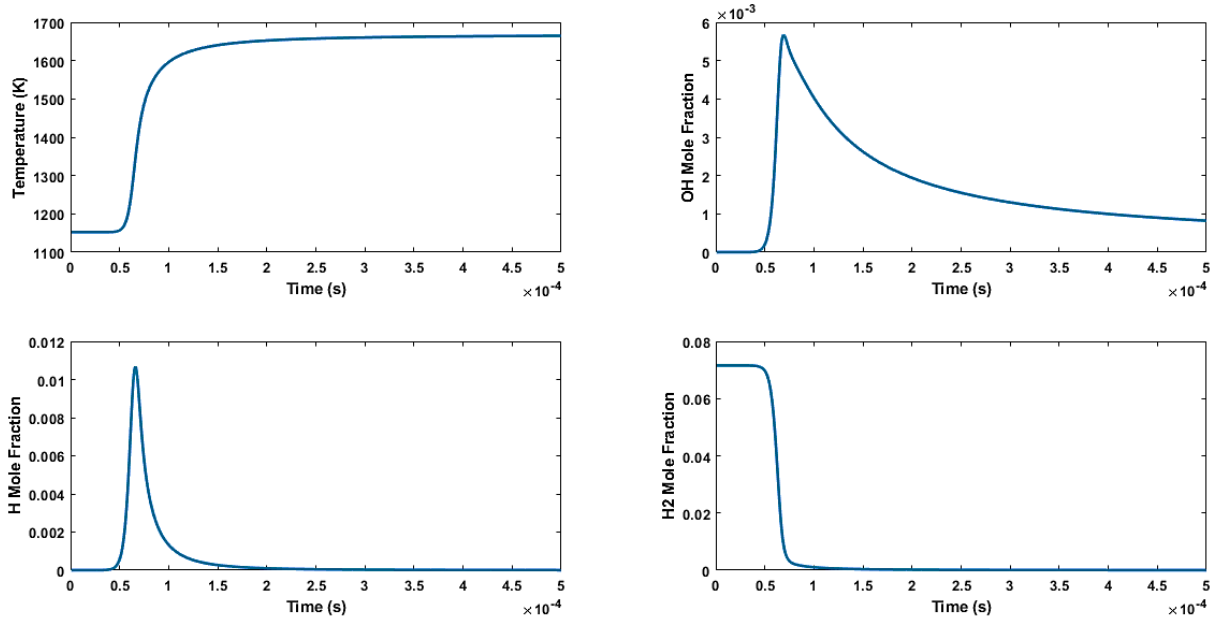


Figure 4.8: Cantera zero-dimensional kinetics simulation results of hydrogen-air combustion with initial conditions of $T_0 = 1152.34\ [K]$, $p_0 = 1.3257\ [MPa]$, and $x_{H_2} = 7.157$ [%], $x_{O_2} = 20.887$ [%]

In the case of pseudo-3D simulation (Wedge), the results were very well matched with the 2D-simulation. And this might be due to considering the same cross-section for the reservoir outlet in both simulations, as explained before. It should be reported that considering the maximum temperature of the simulation, the greatest difference between these two simulations was around $50\ K$. Although the cross-sectional flow rate was exactly the same for both simulations at the reservoir outlet, the flow rate along the centerline of the channel was different. As demonstrated in the following diagrams, the decay of the initial pressure of flow along the centerline was faster in the pseudo-3D simulation which could be the effect of the 3D flow that

takes more energy from the flow to propagates through the channel. Therefore, it can be said that the only difference between these two simulations will be the positions of the phenomena, i.e. shock waves and other changes in the flow field properties will happen in a shorter distance; consequently, if the same time step is considered in both simulations, the results of the 2D-simulation will have higher values than the pseudo-3D one, as shown in the following. Thus, it can be concluded that for finding the effects of the distance between the reservoir outlet and the obstacle, simulation should be carried out in 3 dimensions while in order to just find the temperatures and pressures of the domain without considering the effect of this distance, simulation can be simply performed in 2 dimensions.

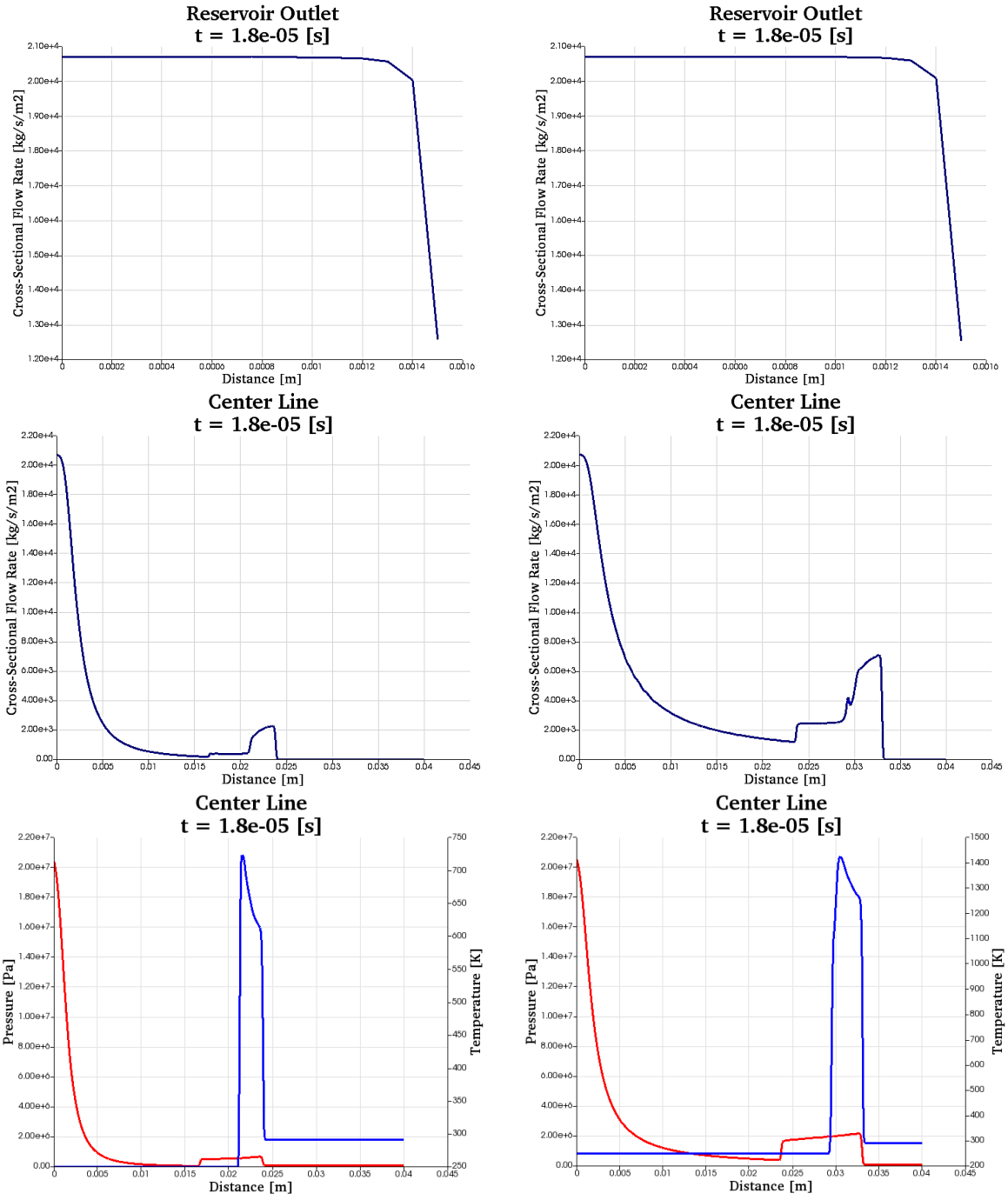
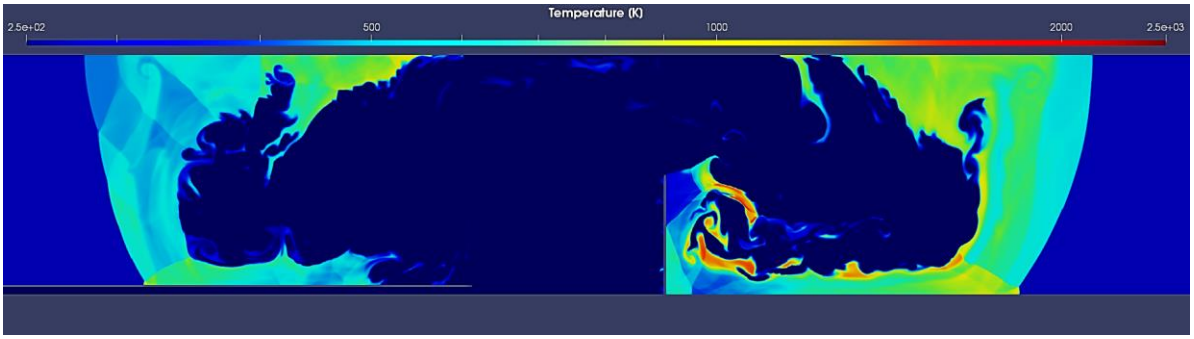


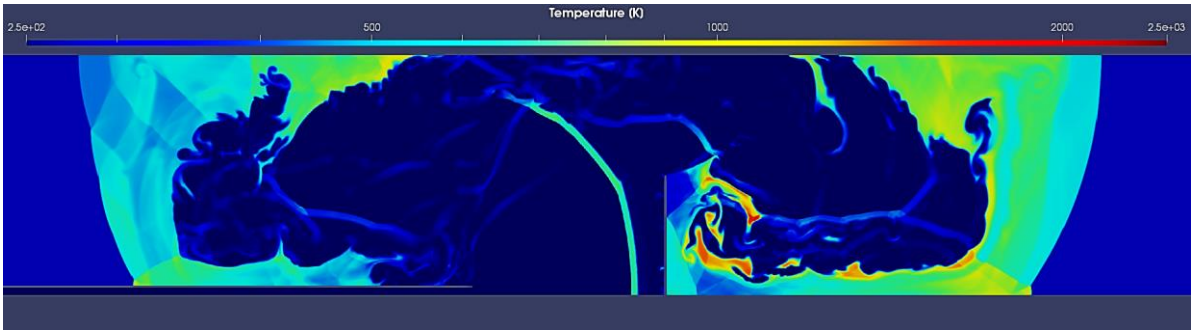
Figure 4.9: Comparing the results of the (left) cylindrical axisymmetric and (right) two-dimensional simulations

After all, there are two main issues with these simulations which will be tried to explain in the following. As mentioned before, due to the size of the computational case (large domain + high grid resolution), in order to use all the power of the computer system, the parallel simulation method was used for this project. Therefore, it was very likely to terminate the simulation for visualizing the results and check them and then continue the same simulation from the last time step. But by using this solver, in the case of having any termination, some discontinuities were generated inside the flow field which was being expanded over time. This issue is depicted in *Figure 4.10*. These temperature contours belong to two consecutive time steps, *Figure 4.10 (a)* is the before termination diagram, and *Figure 4.10 (b)* is the after one. As can be seen some streaks with higher temperatures have been generated at the center of the second figure that are the described discontinuities. To ensure that these continuities are just related to the termination, this was done with other simulations and the same result was achieved. Based on some discussions related to the *rhoCentralFoam*, this may be due to the predicted fluxes at the interfaces. This problem could not be solved by the author and the only solution for that was to try to simulate the entire case, from the beginning up to the desired end time, without any termination.

The second issue is that the minimum temperature of these simulations has reached the limit of the Janaf thermodynamic model of this case which is $250\ K$. This could mean that the real minimum temperature is far below this limit which can affect all the simulation results. To deal with this problem either another set of Janaf coefficients with a wider range of temperature can be used to allow the solver goes below $250\ K$, or another thermodynamic model with a wider operating range should be utilized. Unfortunately, both options are not available, so the only solution left is to implement a new thermodynamic model which will be discussed later in this chapter.



(a)



(b)

Figure 4.10: Temperature contours of two consecutive time steps, (a) before and (b) after the simulation termination

4.6 Validation of the Method and the Shock-Tube Problem

In order to validate this solver and its subsequent settings, the common shock-tube problem was used in this project. In this kind of test, usually, a one-dimensional, frictionless tube with a constant cross-section is considered in which half the tube is filled with a high-pressure fluid called the driver section, and the other half is filled with a low-pressure fluid called the driven section. And it is supposed that there is a membrane between these two sections to prevents them from mixing. At the beginning of the test, this membrane ruptures and the driver section expands which compresses the driven section and this procedure leads to a normal shock wave that propagates through the tube until it reaches the opposite side and reflects. The whole procedure, before the shock reaches the opposite end of the tube, divides the flow field into 4 main sections that can be seen in *Figure 4.11*. After the normal shock hits the downstream end, the 5th region is generated inside the tube which is related to the fluids behind the reflected shock. The privilege of this test is that the expansion behind the normal shock can be studied as an isentropic expansion, so the flow field can be analyzed by the CJ-theory[1] in which the entire phenomenon can be analytically solved without considering the chemical kinetics. And thus, by simulating the same shock-tube problem with a CFD code, the simulation results can be compared to these analytical solutions to ensure that the solver can manage such sharp discontinuities in a proper way. There are several shock-tube simulators on the internet like the Caltech shock and detonation toolbox [40] that can be used to get the analytical solution of a particular shock-tube problem. But the simulator that was utilized in this project is a code written by Are Mjaavatten. This solver depends on rigorous thermodynamic models for hydrogen and air which calculate the thermodynamic properties based on the particular Helmholtz functions for each species [41].

It is worth knowing that in this solver for calculating the properties of the 5th region it is assumed that the normal shock wave reaches the downstream end before it is disturbed by the reflection of the expansion fan from the upstream end [41].

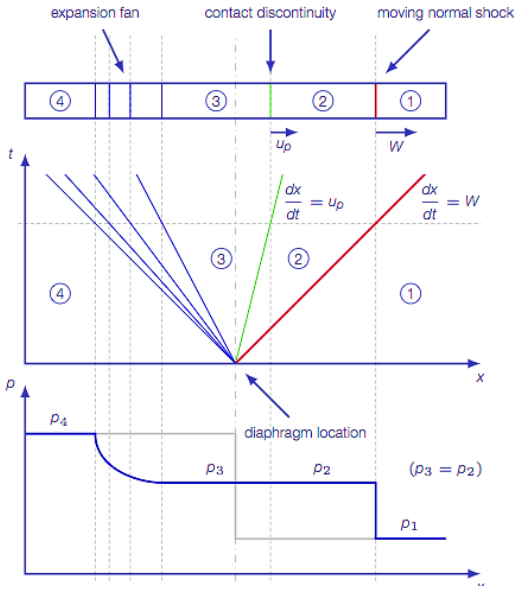


Figure 4.11: (top) Shock-tube problem (middle) Characteristic curves (bottom) Pressure profile [42]

[1] Chapman-Jouguet theory

In the shock-tube problem defined for this section, the driver and driven fluids are respectively, $70 - MPa$ hydrogen and $0.1 - MPa$ air. There are differences between the analytical shock-tube simulator and the simulation done by the $rhoReactingCentralFoam$ solver. In addition to the fact that the former one solves this problem analytically and the latter solves it numerically, the thermodynamic models used to calculate the flow properties of this problem are different between these two, as mentioned before. Besides, the former solver considers air as a pseudo-single species while the latter solver considers air as a mixture of oxygen and nitrogen with specific[1] concentrations of 21% and 79%, respectively. The results of the simulations with these two solvers as well as the shock-tube simulation results by the USN-FLIC code are given in the next section.

4.6.1 Comparison the Results

The Mjaavatten shock-tube simulation results are given below. These simulations were carried out for different initial pressures of driver hydrogen from $1\ MPa$ up to $70\ MPa$ with an initial temperature of $291\ K$. In all cases, the driven air was under the pressure and temperature of $0.1\ MPa$ and $291\ K$, respectively.

As can be observed in *Figure 4.12*, on the next page, the temperature of the compressed air behind the reflected shock increases up to $4729\ K$, while the temperature of the expanded hydrogen drops down to $131.8\ K$. The temperature behind the normal shock, however, remains around $2400\ K$. In addition to these, the pressure behind the reflected shock intensified up to around $42\ MPa$, in the case of having $70 - MPa$ initial hydrogen pressure. As well as these, the same simulations were performed by the *rhoReactingCentralFoam* solver in OF which its results will be discussed in the following.

[1] Based on mass fraction

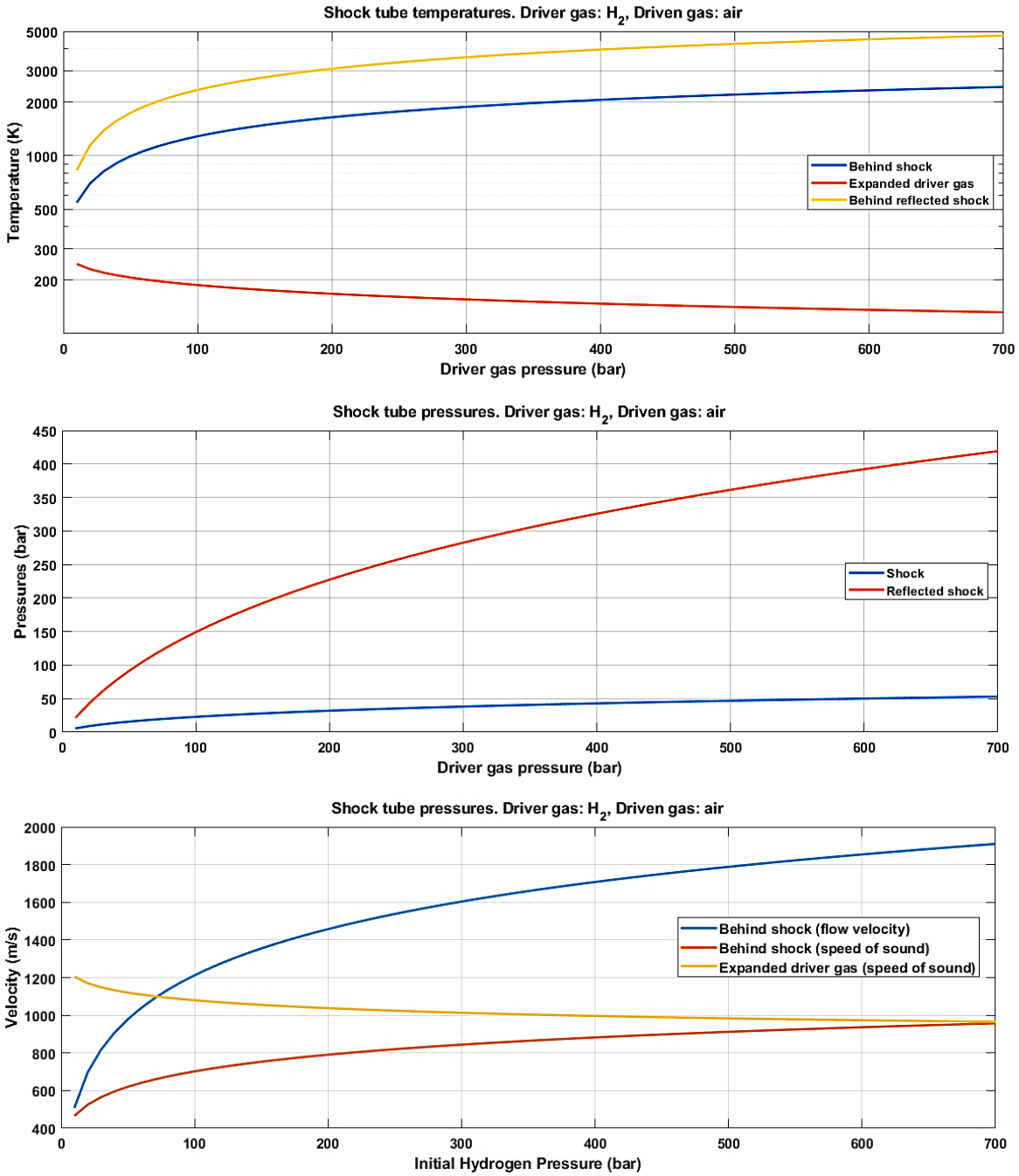


Figure 4.12: The simulation results of the shock-tube simulator (written by A. Mjaavatten)

The OF simulations were carried out with almost the same settings discussed in *Section 4.4*. The difference is that the turbulence was deactivated for this section to consider a laminar flow. Also, the geometry in this section is a one-dimensional tube with a $1\,m$ length. As it was expected according to the discussion about the utilized thermodynamic model in *Section 4.5*, the solver encountered a problem in calculating the temperature for the initial hydrogen pressure equal to $70\ MPa$. This happened after around $0.2\ ms$ of simulation when the normal shock wave reflected from the downstream end and the temperature tended to decrease below $250\ K$ which is the limitation of the utilized Janaf coefficients for thermodynamic model.

If this part is neglected, there was another major problem in all the OF shock-tube simulations even down to 100 MPa hydrogen initial pressure, and that was about passing through the initial membrane position.

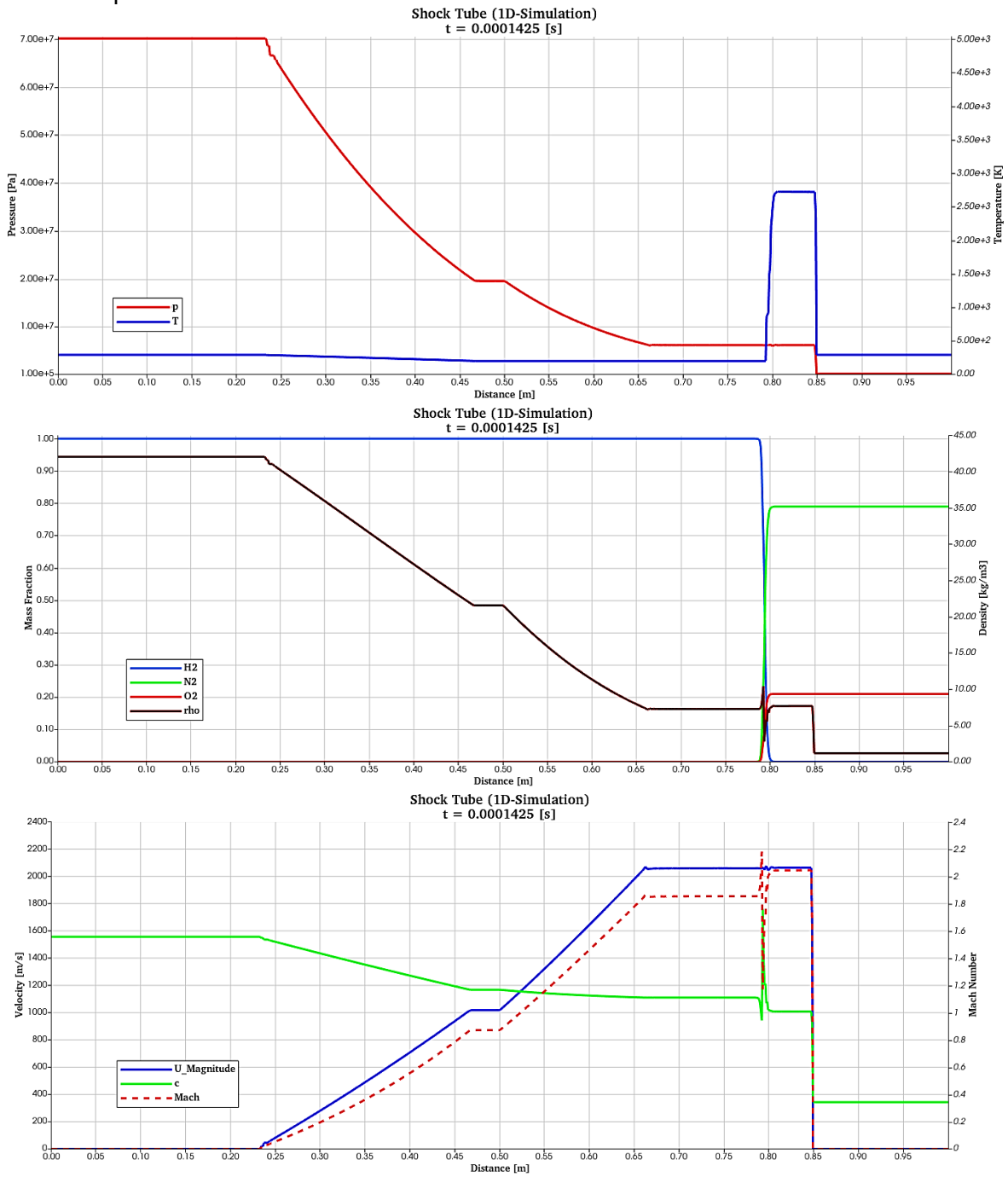


Figure 4.13: *rhoReactingCentralFoam* shock-tube simulation results

As can be observed in *Figure 4.13*, at the center of the shock-tube, all the flow properties flatten out for some distance, like a choked flow, and it is important to note that these flat parts expanded over time and became longer and longer. The reason behind this behavior may be passing through a cross-section with Mach number equal to 1. Although as it is shown, the Mach number of the flow is not exactly equal to 1 at these points, it is just behind this value

and that is what always happens at the front of the initial membrane location. To make sure that other parameters like the grid resolution or the width of the channel (however, it is one-dimensional) are not the reason for this problem, other shock-tube simulations with different such parameters were carried out, but in all the cases, this problem was discovered. Therefore, it should be considered that the solver may have some problems in dealing with transonic flows.

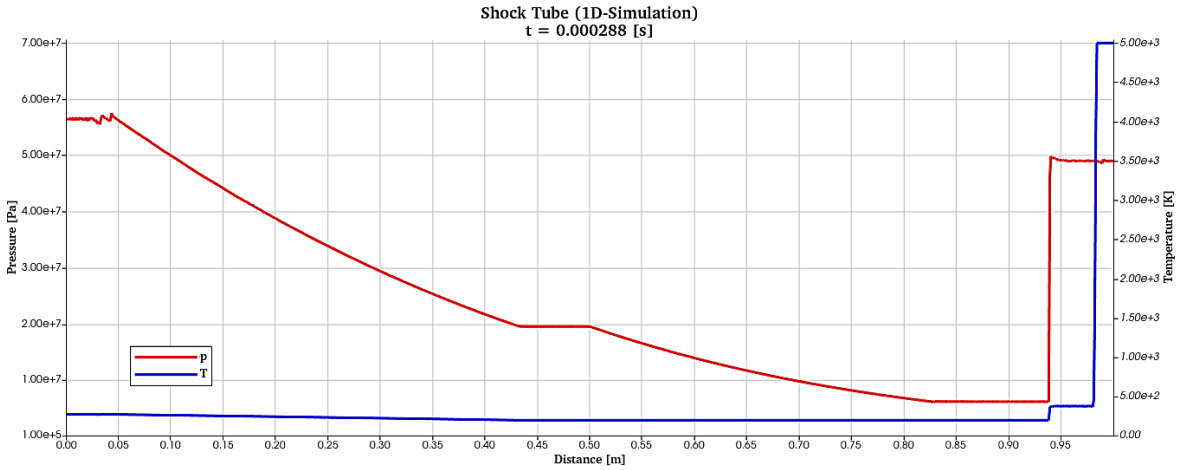


Figure 4.14: Pressure and temperature profile of the OF shock-tube simulation after the normal shock hits the downstream end of the tube

All in all, it is important to add that the predicted pressures, temperatures, and velocities in the OF shock-tube simulation are almost the same as the Mjaavatten shock-tube results, despite all the mentioned problems. Even behind the reflected shock wave, as shown in *Figure 4.14*, the maximum of the temperature and pressure stayed on track. And this point is important since the results of the USN_FLIC code shock-tube simulation with the same initial conditions are far away from these values of the flow properties. Thus, the results of the USN_FLIC code simulation of the main scenario will also not be reliable anymore.

The results of the USN_FLIC code shock-tube simulation for two different time steps, corresponding to the time steps of the demonstrated OF simulation results, are shown in *Figure 4.15 (a)* and *(b)*. Although these results do not have the same problems as the OF simulation, the values of the flow properties are overestimated. As can be seen, the temperature of the compressed air before the normal shock reaches the end is around $3000\ K$ which have a $500\ K$ difference with above simulations, while after reflection this difference becomes even much more since the temperature of the compressed air reaches $7000\ K$.

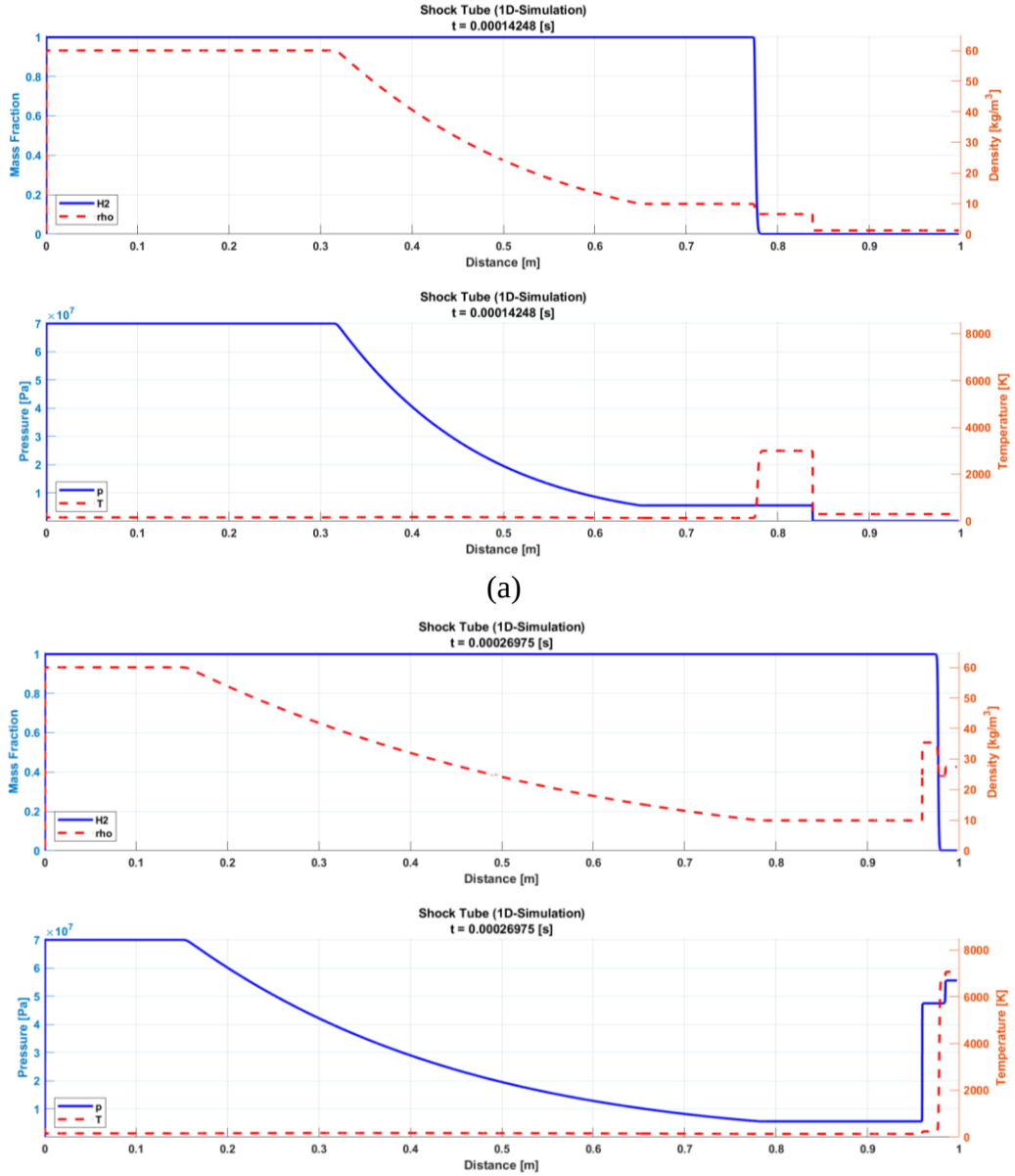


Figure 4.15: USN_FLIC code shock-tube simulation results (a) before the normal shock reaches the downstream end (b) after the normal shock reaches the downstream end

According to all these discussions, it can be concluded that if the problems of the OF solver can be fixed, then it could be a proper option to simulate the original case. In order to stretch the limits of the thermodynamic model utilized in *rhoReactingCentralFoam* solver, new thermodynamic and transport models were tried to implement in OF v.7 which will be explained in the next section.

4.7 Implementation of the New Thermodynamic and Transport Models

Regarding the thermodynamic and transport models, there are plenty of equations and approaches to estimating the thermophysical properties of individual species. In an extreme case, each of these species can have its own particular equation to calculate its thermodynamic and transport properties as accurately as possible. But in this project, in order to have a general model for a wide range of temperatures, in the beginning, it was tried to implement the so-called NASA-7 and NASA-9 equations [43], [44].

4.7.1 nasa9Poly Thermodynamic Model

There are equations and their corresponding coefficients published by NASA for calculating the thermodynamic properties of species. These equations in general are in the same form as the polynomials used in the Janaf thermodynamic model. But for simulating the properties in a wider range of temperatures, the NASA polynomials use more terms for calculations. The total number of coefficients in these equations is 9 and that is why this model is called *nasa9Poly* thermodynamic model. The equations are as follows [44]:

$$\begin{cases} \dfrac{C_p^0(T)}{R} = a_1T^{-2} + a_2T^{-1} + a_3 + a_4T + a_5T^2 + a_6T^3 + a_7T^4 \\ \dfrac{H^0(T)}{RT} = -a_1T^{-2} + a_2 \ln T\,/T + a_3 + a_4T/2 + a_5T^2/3 + a_6T^3/4 + a_7T^4/5 + b_1/T \\ \dfrac{S^0(T)}{R} = -a_1T^{-2}/2 - a_2T^{-1} + a_3 \ln T + a_4T + a_5T^2/2 + a_6T^3/3 + a_7T^4/4 + b_2 \end{cases} \quad (4.20)$$

In these relations, $C_p^0(T)$, $H^0(T)$, and $S^0(T)$ are the ideal-gas part of, respectively, the specific heat capacity at constant pressure, the enthalpy, and the entropy. The advantage of using this model is that the ideal-gas part of the thermodynamic properties of most species can be calculated from $200\ K$ up to $20{,}000\ K$. And for doing so 3 sets of 9 coefficients for 3 different ranges of temperatures will be used. These ranges are usually categorized to $200 - 1000\ K$, $1000 - 6000\ K$, and finally $6000 - 20{,}000\ K$. These specific NASA-9 coefficients are tabulated in [44]. Besides, the detailed C++ dictionary implemented for this thermodynamic model in order to use for OF simulations is given in *Appendix G*.

4.7.2 nasaPoly Transport Model

In addition to the discussed thermodynamic polynomials, there is another set of equations and their corresponding coefficients for calculating the transport properties of individual species, i.e. the diffusion coefficients. These equations and coefficients, also, published by NASA and their estimations are much more accurate than the utilized Sutherland transport model. The temperature ranges of these polynomials, however, are not as wide as the thermodynamic model. Therefore, they just need 2 sets of coefficients, usually from $200 - 1000\ K$, and from $1000 - 5000\ K$. These equations and their coefficients are related to what is called the NASA-7 thermodynamic model which is the older version of NASA-9 with polynomials that only need 7 coefficients. But, the transport polynomials of NASA-7 only use 4 coefficients in order to calculate the transport properties and that is why this implemented model is just called the *nasaPoly* transport model. The equations of this model are as follows[43]:

$$\left.\begin{matrix}\ln \mu \\ \ln k\end{matrix}\right\} = A \ln T + \frac{B}{T} + \frac{C}{T^2} + D \tag{4.21}$$

The sets of coefficients for calculating dynamic viscosity (μ) and thermal conductivity (k) will be different. The detailed C++ dictionary implemented for this transport model in order to use for OF simulations is given in *Appendix H*.

It is also important to note that a new thermophysical model based on these two new thermodynamic and transport models had to be implemented in OF just like what was done in *Section 4.3* to make them functional.

4.8 Generating New Sets of Coefficients for the New Models

As described above, the implemented thermodynamic and transport models have much wider operational ranges as well as more accurate estimation results. Nevertheless, their temperature ranges still are not sufficient, particularly for simulating very low hydrogen temperature. Therefore, to cope with this issue, it was tried to find new sets of coefficients for the new thermodynamic and transport models for covering temperatures below $200\ K$.

In the case of the *nasa9Poly* thermodynamic model, the idea was to use the feature of this model that can have 3 different sets of coefficients for 3 consecutive ranges of temperature. As discussed in the previous section, the original category is $[200 - 1000]\ K$, $[1000 - 6000]\ K$, and finally $[6000 - 20000]\ K$. But even for the shock-tube problem, according to the mentioned results, the maximum temperature does not go above $5000\ K$. So, in fact, the last temperature interval is not useful for this project and these ranges can be shifted like this: $[boiling\ temperature - 300]\ K$, $[300 - 1000]\ K$, and finally $[1000 - 6000]\ K$. However, for using this new category, a new set of coefficients should be derived for the first temperature interval (from the boiling point up to $300\ K$). This new set should be consistent with the other coefficients and can be used in *Equation (4.20)*.

In the other case, the *nasaPoly* transport model, the idea was to just try to stretch the lower interval down to the boiling point. Because this transport model, despite the thermodynamic model, can only use 2 sets of coefficients for 2 consecutive ranges of temperatures. Therefore, the transport model intervals should be shifted like this: $[boiling\ temperature - 1000]\ K$, and $[1000 - 6000]\ K$. These sets, also, should be consistent with *Equation (4.21)*.

The reason behind choosing the boiling point as the lower limit of all these intervals is that below this temperature the phase of the species will change so there would be a large jump in the thermophysical properties of species before and after this temperature.

After all, in order to generate the discussed coefficients, 3 databases were written in Python for hydrogen, oxygen, and nitrogen. And the goal was to compare the species properties from different available datasets to make sure that the new sets of coefficients, which are intended to derive, are conforming with the real values. The compared datasets for each of these species are as follows:

- Hydrogen:
    1. NASA-7 [43]
    2. NASA-9 [44]
    3. JANAF Thermochemical Tables [45]

4. U.S. National Bureau of Standard (NBS) [27]
5. National Institute of Standards and Technology (NIST) [46]
6. Fundamental EOS for Hydrogen (Leachman EOS) [47]
7. OpenFOAM v.7 (Janaf and Sutherland models)

- Oxygen:
  1. NASA-7 [43]
  2. NASA-9 [44]
  3. JANAF Thermochemical Tables [45]
  4. U.S. National Bureau of Standard (NBS) [48]
  5. National Institute of Standards and Technology (NIST) [46]
  6. Thermodynamic Functions for Oxygen (Woolly EOS) [49]
  7. OpenFOAM v.7 (Janaf and Sutherland models)
- Nitrogen:
  1. NASA-7 [43]
  2. NASA-9 [44]
  3. JANAF Thermochemical Tables [45]
  4. U.S. National Bureau of Standard (NBS) [48]
  5. National Institute of Standards and Technology (NIST) [46]
  6. Reference EOS for Nitrogen (Span EOS) [50]
  7. OpenFOAM v.7 (Janaf and Sutherland models)

As an example of all these data and relations, the values of the specific heat capacity at constant pressure (C_p), and the dynamic viscosity (μ) for hydrogen, based on the mentioned references, are shown in *Figure 4.16* and *Figure 4.17*.

At last, for generating new sets of coefficients, the focus was on the particular equations of state for each of these species (Leachman EOS for H_2, Woolly EOS for O_2, Span EOS for N_2). Because these equations can model low temperatures very well. Therefore, a program was written in Python in order to generate these coefficients by using the method of least squares in a way that the derived coefficients are not only fit into the coefficients of the polynomials of the new thermophysical models (*Equation (4.20)*, and *Equation (4.21)*) but also consistent with the discussed specific equations of state.

As can be seen in *Figure 4.16* and *Figure 4.17*, the derived coefficients are very well overlapped with the desired models, even at low temperatures down to the boiling point. Therefore, it seems that they are reliable and ready to utilize as the coefficients of the new thermodynamic and transport models which will be done in the next section for simulating the same shock-tube problem with the new thermophysical model.

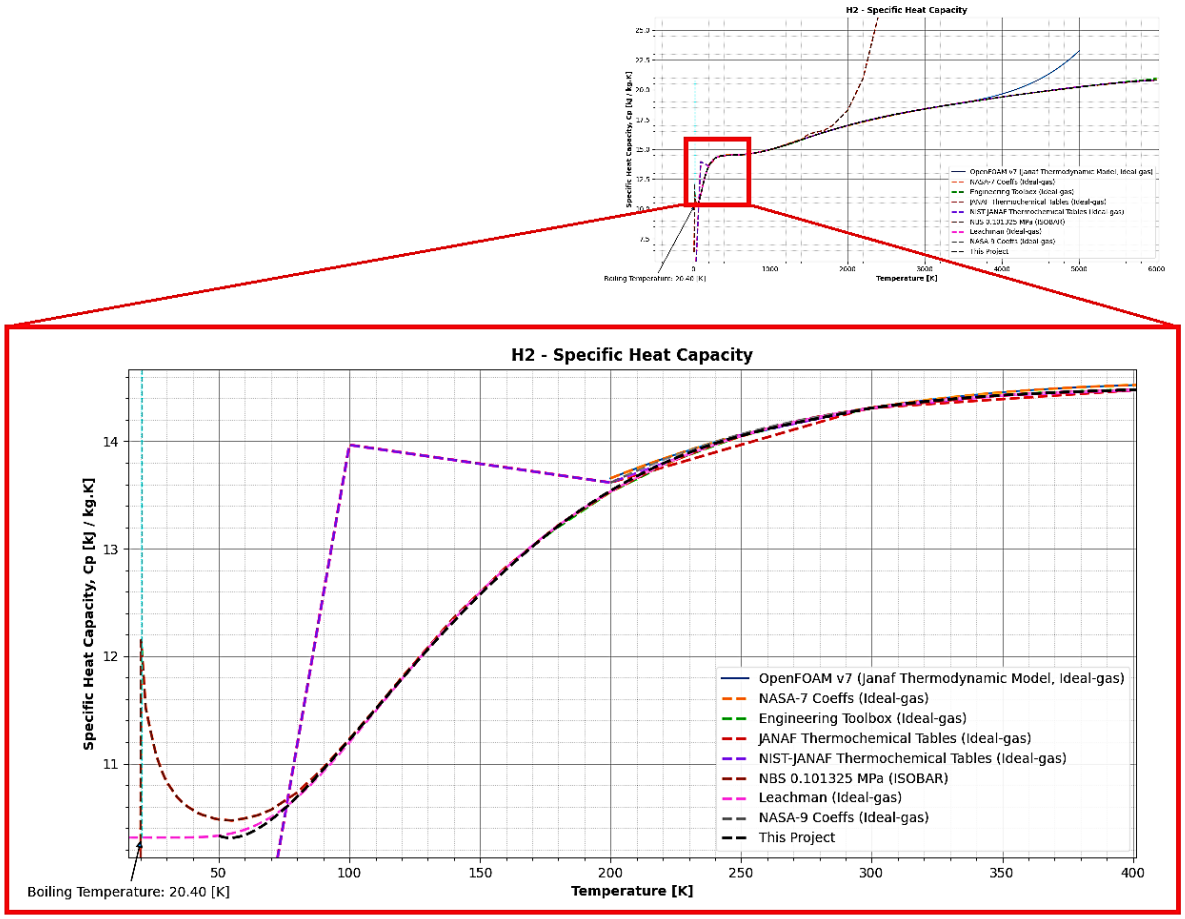


Figure 4.16: Hydrogen specific heat capacity profile from different datasets

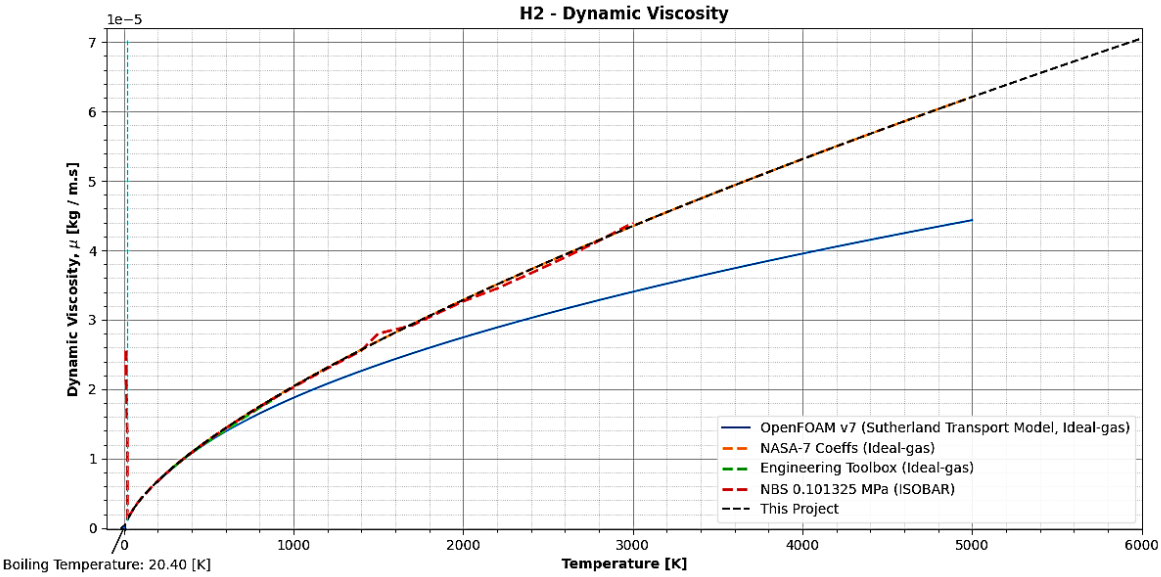
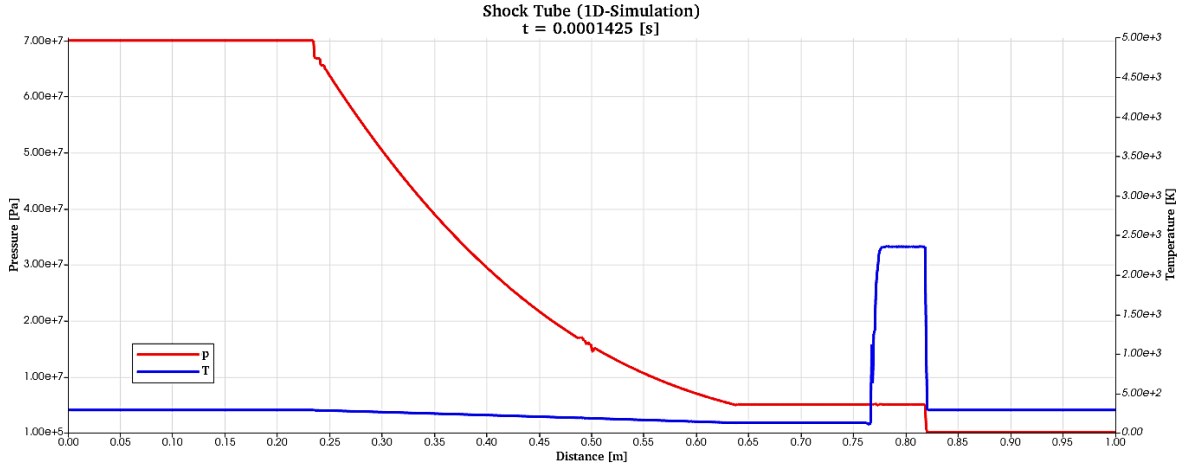


Figure 4.17: Hydrogen dynamic viscosity profile from different datasets

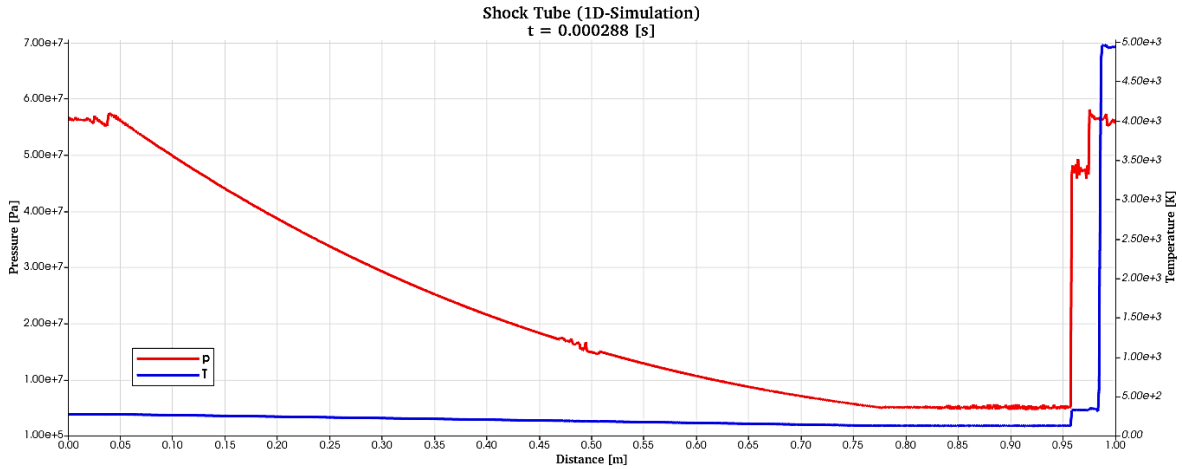
4.9 Shock-Tube Simulation with the New Thermophysical Model

For the shock-tube simulation with the new thermophysical model containing the new *nasa9Poly* thermodynamic model and *nasaPoly* transport model as well as the new generated sets of coefficients, the general settings are the same as before; and the only difference is the usage of these new models. The detailed coefficients in the *thermo.compressibleGas* dictionary for this simulation is given in *Appendix I*.

As an illustration, the new shock-tube simulation results are shown in *Figure 4.18*. In order to be comparable with previous results, the same time steps as in *Figure 4.13*, and *Figure 4.14* are chosen. As can be observed the discussed flat lines at the center of the tube converts to some noises and despite the previous simulation, the solver can roughly keep track of the isentropic expansion. The temperature of the compressed air before and after the normal shock reflection is still at the same range as the results of the Mjaavatten shock-tube solver. However, there are two issues with this simulation. first, the flow properties especially after the normal shock reflection are noisier and more unstable than the results of the previous simulation. As well as this, the pressure profile after the reflection is not as accurate as before.



(a)



(b)

Figure 4.18: Pressure and Temperature profiles of the OF shock-tube simulation results with the new thermophysical model (a) before normal shock reaches the downstream end (b) after it hits the end

Although these shock-tube simulations are only one-dimensional and there is no such a thing as constriction of the cross-section, still the cross-sectional flow rate [$\frac{kg}{m^2 \cdot s}$] of these simulations can be a good criterion to study the choked flow. In *Figure 4.19* the flow rate profile of the OF shock-tube simulations with the new and old thermophysical models are drawn. As can be seen, the new thermophysical model (the blue line) can smoothly pass through the local maximum, while the old one (the red line) as it goes to the maximum, at some point it could not increase any more, and for some distance, the flow rate remains constant. This, finally, leads to different predictions for the locations of the contact surface (between hydrogen, oxygen, and nitrogen) and normal shock wave.

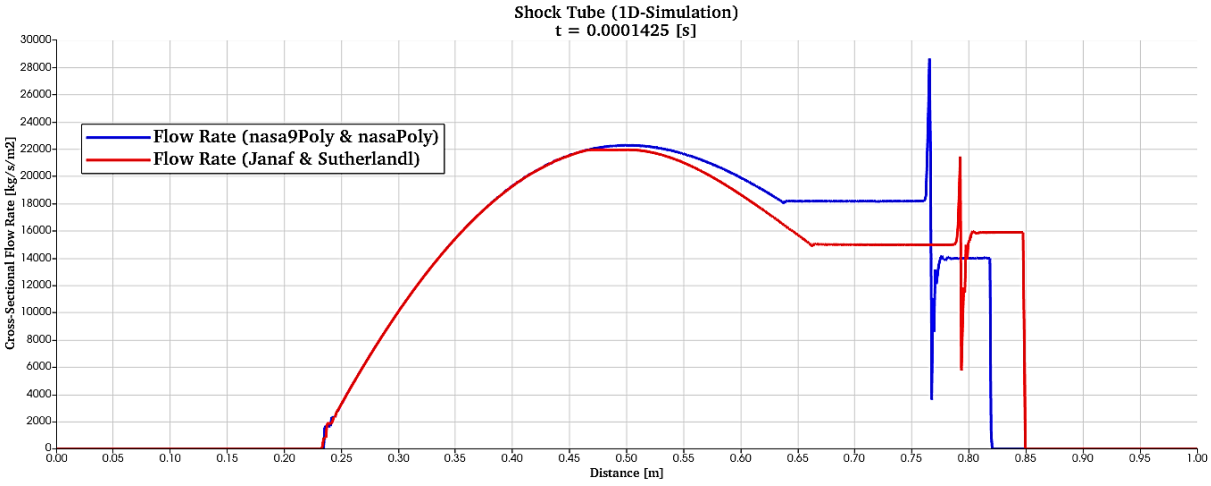


Figure 4.19: Cross-sectional flow rate of the OF simulations with two different thermophysical models

The main results of all these 4 shock-tube simulations that have been so far considered are tabulated in *Table 4.3*. All in all, apparently, the combination of the *rhoReactingCentralFoam* solver and the new thermophysical model can have more accurate simulation results, although it seems more unstable than the previous OF shock-tube simulation. Thus, the next step would be checking this new method for simulating the main scenario which will be discussed in the next section.

Table 4.3: Comparison between the results of the shock-tube simulations

Shock-Tube Simulation / Flow Property	Mjaavatten Solver	USN-FLIC Code	OpenFOAM (Janaf + Sutherland)	OpenFOAM (nasa9Poly + nasaPoly)
Temperature behind the normal shock [K]	2434	3000	2720	2360
Temperature behind the reflected shock [K]	4729	7070	5000 (upper limit)	4950
Expanded driver fluid temperature [K]	131.8	127	250 (lower limit)	120
Normal shock pressure [MPa]	5.295	5.602	6.088	5.1
Reflected shock pressure [MPa]	41.91	55.57	49	≈ 50 (large fluctuation)

4.10 Final Simulation Results and Discussion

In the first attempt for simulating the main scenario by using the new thermodynamic and transport models along with the newly generated coefficients, exactly the same settings as the previous main simulation was used; except for utilizing this new thermophysical model. Unfortunately, the solution was diverged after just simulating 1.4 μs with the *Floating point exception* error.

After this, to avoid getting this error and to have a more stable solution the *relaxationFactors* for all equations were reduced. But even this change could not help the solver to pass the initial steps. Since it seemed that the problem is related to the hydrogen output (from the reservoir), so another strategy was chosen to deal with the initial extremely large gradient. The approach was to change the shape of the reservoir output from a rectangular one to a conical one. As can be seen in *Figure 4.20*, the reservoir output will not have a sharp break and can lead the flow more smoothly. Fortunately, this method was successful and the solver could pass the initial time steps.

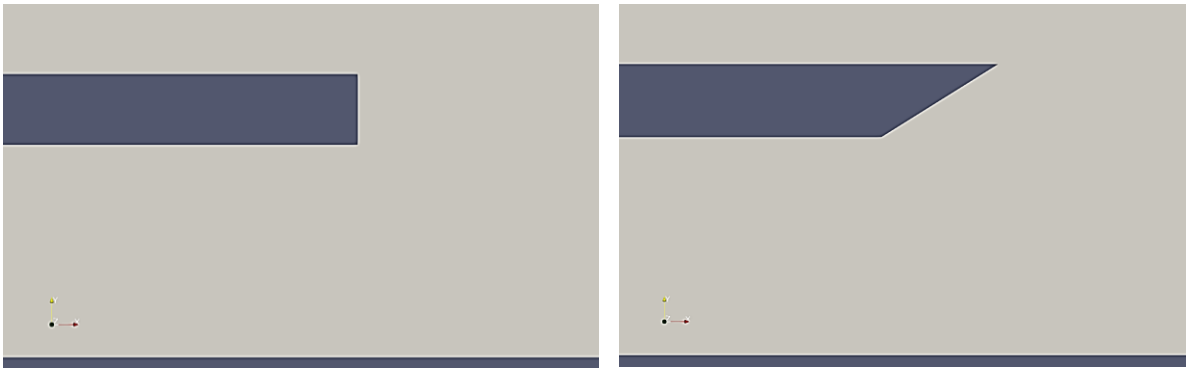
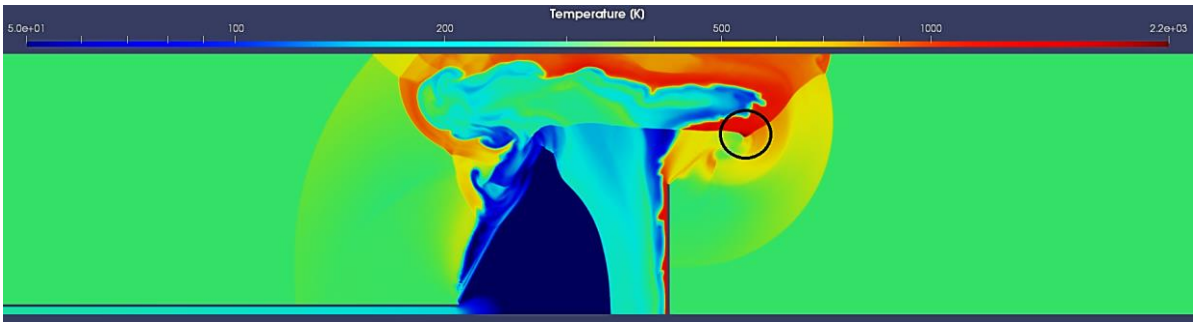


Figure 4.20: Changing the reservoir output (reservoirWall) from a rectangular shape into a conical one

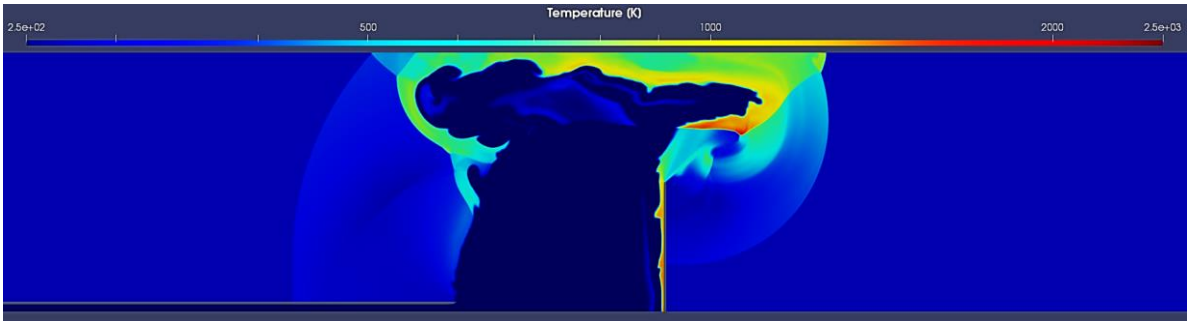
But, unfortunately, this time after around 87 μs, the solver reaches a point that cannot find the accurate temperature (tolerance $< 1e-04$) by less than 100 iterations. So, the following error was popped up: *Maximum number of iterations exceeded: 100*. This error is related to the *thermo* dictionary in which the solver tries to find the actual temperature by guessing an initial temperature and then after calculating the internal energy (or enthalpy in other cases) and consequently specific heat capacity at constant volume (or constant pressure in other cases), compute a new temperature and if the difference between the initial one and the calculated one was less than $1e-04$, then accept that temperature. The maximum number of this kind of iteration is set to be 100. So, if a divergent solution is triggered by a reason, after some point the solver could not calculate the accurate temperature in less than 100 iterations. And that is what happened in this case.

According to the boundaries of the hydrogen cloud, or more precisely, to the edges of the developed flow, at the last step of the above simulation, it seemed that the reason behind this divergence is the grading that was used in generating the mesh to have a less number of total cells. Therefore, it was decided to eliminate this grading and generate a new uniform structured mesh with a grid resolution of 100 μm. As far as the time constraints of this project have allowed simulating this final case, fortunately, there was no problem during the simulation. Nevertheless, because this solver and its settings showed a lot of instabilities in simulations, the author could not be sure that if there was more time for running the simulation, no error would occur.

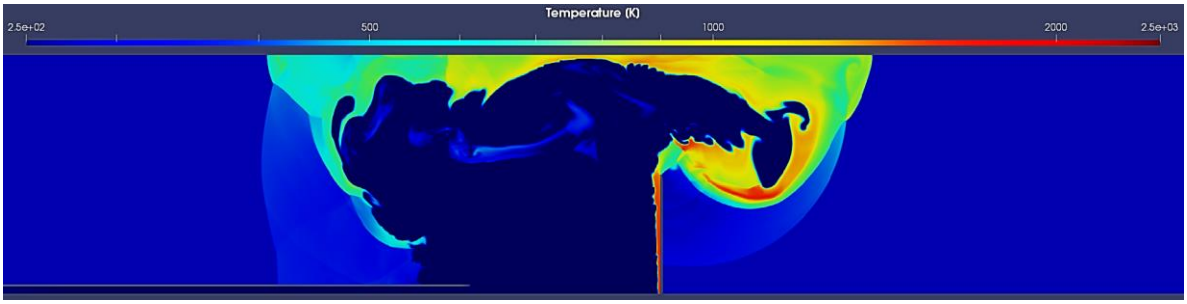
After all, the total number of cells in the newly generated mesh was 7,475,050. Therefore, not only the simulation itself but also even visualizing the results took much more time than before. During the run time, totally, $1.03e-04\ s$ could be simulated. By using the *minMaxMagnitude* post-processing function, it was revealed that the maximum temperature has happened at $7.58e-05\ s$, and its value reached $2214.25\ K$. But in this simulation, despite all the previous simulations the temperature reached its maximum after the reflected shock from the *Top* combined with the rarefaction shock from the *Obstacle* which is shown by a black circle in *Figure 4.21 (a)*. Also as can be seen in this figure, temperature values reach the lower limit of the utilized thermophysical model which is $50\ K$; this is not a good sign for a simulation, since most probably this means that the actual temperature of the expanded hydrogen is different than $50\ K$ (not necessarily colder).



(a)



(b)



(c)

Figure 4.21: Temperature profile at $t = 7.6e-5\ [s]$ in (a) final simulation (b) final simulation with adjusted temperature interval (c) initial 2D-simulation

In order to better visualizing the comparison between the results of this simulation and the initial 2D-simulation, the temperature range of *Figure 4.21 (a)* is changed to the interval of the temperature of the initial simulation, $[250 - 2500]\ K$, which is shown in *Figure 4.21 (b)*. And finally, at the bottom of this figure, *Figure 4.21 (c)*, the result of the initial 2D-simulation at the same time is shown. As can be observed from these temperature contours, hydrogen flow

in the initial simulation has propagated more than the final one. The reason behind this behavior could be the noticeable difference between the estimated viscosity by the *nasaPoly* and *Sutherland* transport models. The former model, as can be seen in *Figure 4.17*, calculates higher values in every temperature in comparison to the latter one.

To have a general overview from the flow structure of this simulation, the flow properties at the last simulated time step is shown in the following diagrams. As can be seen in the above diagram, one of the characteristics of this simulation is that the edges of hydrogen cloud are not as wrinkled as the initial simulation which could be again due to the viscosity difference between these two simulations. Also, it is obvious that changing the shape of the reservoir outlet made a difference in the flow field, especially, in front of the outlet.

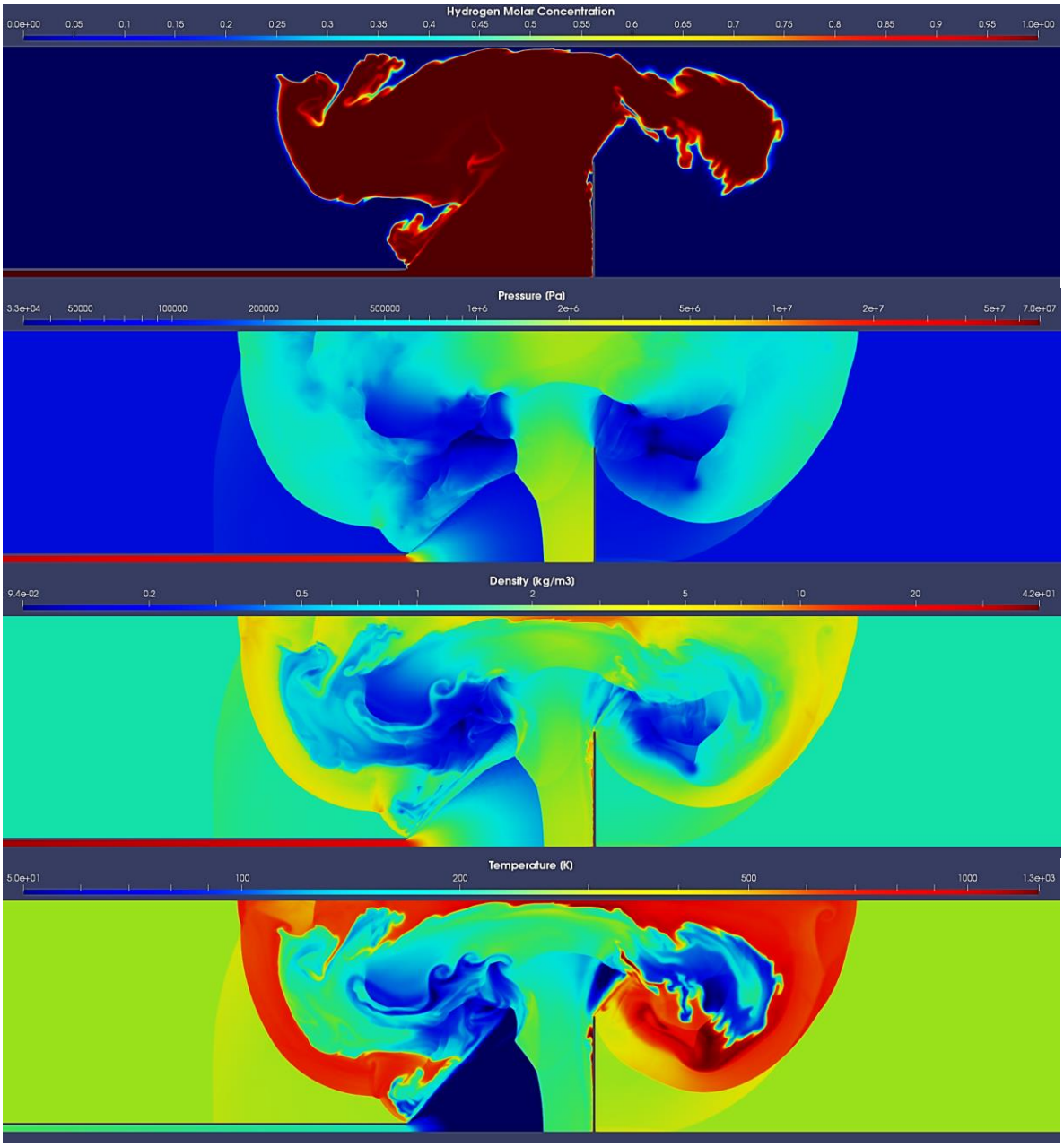


Figure 4.22: Flow properties of the OF final 2D-simulation at $t = 1.03e-04\ [s]$

Comparing the flow properties along the center line, in addition to the fact that the positions of the shock waves in the initial simulation is further than the final one away from the reservoir outlet, reveals one more point. This point is more obvious in the below diagrams of *Figure 4.23*, as can be seen in the initial simulation at the first discontinuity in the pressure profile, the temperature remained untouched while in the final simulation this is accompanied with a temperature jump.

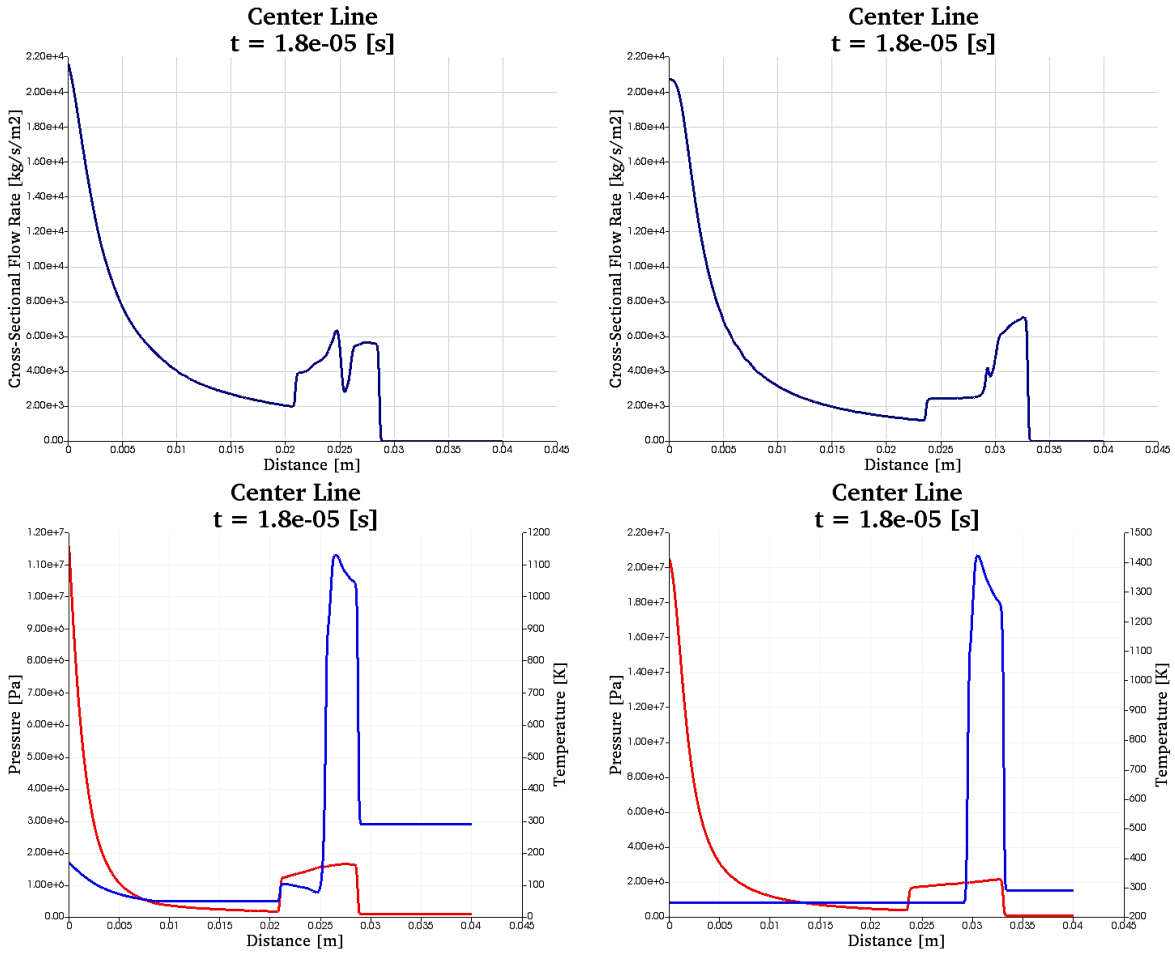


Figure 4.23: Comparing the results of the (left) final and (right) initial OF simulations

At last, considering the stream path of the flow, just like before, shows a large eddy behind the *Obstacle* with a high temperature that is demonstrated in *Figure 4.24*.

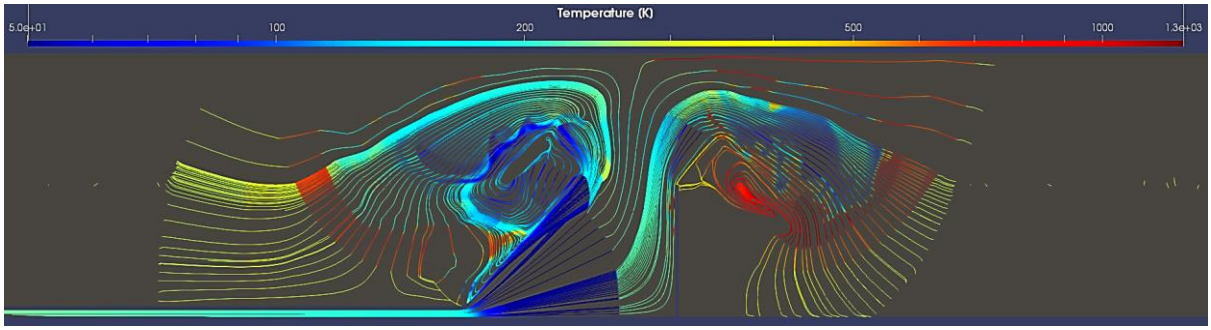


Figure 4.24: Stream lines at $t = 1.03e - 04\ s$ in the final OF simulation

5 Conclusion

Nowadays, hydrogen is considered as one of the available, sustainable solutions to reduce the carbon footprint from industrial and even individual and daily activities. Therefore, conducting studies like the present project is crucial for acquiring a sufficient understanding of hydrogen behavior and its potential hazards under different circumstances. One of the major risks in utilizing hydrogen is its spontaneous ignition which can be occurred during any kind of hydrogen release.

In the present project, the main goal was investigating the relatively long-term effects of the downstream obstructions on this auto-ignition phenomenon in a high-pressure hydrogen release into the air. The initial hydrogen pressure considered for this study was $70\ MPa$ which is among the highest pressures used in CGH2 reservoirs.

This investigation was carried out with 2 different sets of CFD codes, the USN-FLIC code, and OpenFOAM. In the case of OpenFOAM, due to the lack of a proper built-in solver, a recently developed solver was used, and for improving its abilities new sets of thermophysical models were implemented. Also, it was tried to demonstrate the strengths and weaknesses of each of these solvers in simulating this extreme case.

On the basis of the results and their validations, it can be concluded that the USN-FLIC code, with presented settings, is not able to predict reasonable flow properties for the scenario of two-dimensional $70-MPa$ hydrogen release into the air. Although the general flow structures, such as shock waves, reflections, etc. were simulated successfully with this code without having spurious oscillations, the flow properties, like temperature, were overestimated by this code. The main reason could be using the ideal-gas equation of state and neglecting the compressibility effects.

In the USN-FLIC code simulation case, although the simulation itself was faster than other cases in this study, the size of the generated data was too large even with compressing which made it difficult to transfer and to establish the post-processing section.

Regarding the OpenFOAM simulations, the combination of *rhoReactingCentralFoam* solver along with the first thermophysical model (including the Janaf thermodynamic model, Sutherland transport model, and Peng-Robinson equation of state) generated reasonable flow properties for the main scenario, although further investigations into the shock-tube validation test revealed that this solver and thermophysical model are not capable of smoothly passing through a sonic condition and keep the isentropic expansion curve in shape. Besides, in both the main scenario and shock-tube simulations, the temperature of the flow reached the limitations of temperature ranges in the thermodynamic model. Therefore, the results of the main scenario simulation with this OpenFOAM setting, also, could not be reliable. However, it should be mentioned that this combination of *rhoReactingCentralFoam* solver and the described thermophysical model, produced very stable solutions which should be considered for further studies.

Furthermore, it was shown that two-dimensional cylindrical axisymmetric simulation affects the flow propagation in a way that every phenomenon happens as same as the two-dimensional simulation but at a shorter distance. Thus, for investigating the distance effect of the downstream obstructions the simulation should be done in three-dimensions.

According to the results of the OpenFOAM simulation with *rhoReactingCentralFoam* solver and the second thermophysical model (including the implemented *nasa9Poly* thermodynamic model, *nasaPoly* transport model, and the Peng-Robinson equation of state), although this setting could relatively smoothly pass through the sonic condition and had the best conformity to the Mjaavatten shock-tube solver among the utilized solvers in this project, its solutions produced more instabilities in comparison to the previous OpenFOAM simulation. These instabilities which showed themselves as a noisy solution in the shock-tube simulation, especially after the reflection, led to several divergences in the main scenario simulation and made it very difficult to run the simulation for a long time. However, in the last case, at least as far as the time constraints of this project had allowed running the simulation, no problem occurred.

In general, the latter OpenFOAM simulation setting predicted lower temperatures for the flow field, and despite the former one the maximum temperature did not occur at the beginning of the hydrogen release, instead, it happened due to the interaction of the shock waves reflected from the downstream obstructions. At last, it should be mentioned that the OpenFOAM simulations were much more time consuming than the USN-FLIC code, and also visualizing the results needed a lot of computer power and time.

All in all, simulating such a high-pressure hydrogen release in a large domain and with a high grid resolution needs not only a stable, precise solver but also a lot of time and computer power. According to the general flow structure, the obtained results proved that there are potentially dangerous regions inside the domain that are formed due to the existence of the obstacles and confinement. However, because each of these solvers and methods had its own problems and also estimated different flow properties, the hazard rate of this scenario cannot be proved with high certainty.

6 Recommendations

As it was discussed in the previous chapters, there are several uncertainties over the results of the simulations and one of the reasons is the very high initial hydrogen pressure in this project which also leads to a very wide temperature range in the flow field. To deal with these issues and develop this study, some tips will be recommended in the following:

1. **OpenFOAM +:** Two main communities develop the OpenFOAM code, the OpenFOAM Foundation[1] , and the ESI group[2]. Although initially OF was released by the former group, these days the OF versions of the latter group provide more flexibility and options for the user. Thus, there are lots of solver-developers who are currently working on this version. So, changing from the former version of OF to the latter one could give more accessibility to the new solvers.
2. **New Solver:** As the author was working on a new solver based on the combination of the PIMPLE algorithm and *rhoCentralFoam* solver, which was unfortunately not prepared for this report, it seems that using such an algorithm improves the calculations of the pressure. Which may solve the sonic problem of the *rhoReactingCentralFoam* solver and improve the stability of the solution.
3. **New Thermophysical Model:** Implementing discontinuous thermophysical functions for species in order to calculate the flow properties more accurately especially at extremely low and high temperatures can improve the results of these simulations. But it should be taken into account that considering these temperature ranges is corresponding to dealing with changing phase and also dissociation of the species. Therefore, in general, adding the chemical reactions to the simulations will definitely create great impacts on the results.
4. **C++ Cantera Toolbox:** Using the Cantera toolbox along with the OpenFOAM software and combining them in a new solver for dealing with the chemical kinetics aspect of the simulation could improve solving the species transport equations and giving a better estimation from species concentrations.
5. **Geometry:** In this study, the effects of some parameters like the nozzle diameter or the distance between the hydrogen output and the obstacle did not investigate. But these parameters will certainly play an important role in forming the potentially hazardous regions. Therefore, considering them as variables would be a major step. However, it will definitely consume a lot of time to simulate.
6. **Experimental Test:** Conducting limited experimental studies for the same scenarios will provide excellent sources for validating the results of the simulations.

[1] www.openfoam.org

[2] www.openfoam.com

7 References

[1] J. Gummer and S. Hawksworth, “Spontaneous ignition of hydrogen / Literature Review,” Health and Safety Laboratory, Buxton, 2008. [Online]. Available: https://www.hse.gov.uk/research/rrpdf/rr615.pdf.

[2] G. R. Astbury and S. J. Hawksworth, “Spontaneous ignition of hydrogen leaks,” presented at the 1st International Conference on Hydrogen Safety, Pisa, Italy, 2005.

[3] U.S. Department of Energy, “Best Practices Overview / Hydrogen Compared with Other Fuels,” *Hydrogen Tools (H2tools.org)*. https://h2tools.org/bestpractices/hydrogen-compared-other-fuels.

[4] A. Bain, *Sourcebook for hydrogen applications*. Tisec Inc. Montreal Canada: Hydrogen Research Institute and National Renewable Energy Laboratory, 1998.

[5] Office of Energy Efficiency & Renewable Energy, “Hydrogen Storage,” *U.S Department of Energy*. https://www.energy.gov/eere/fuelcells/hydrogen-storage.

[6] P. M. Hansen, D. Bjerketvedt, and W. Rondeel, “EET3210-1 20H Energy Technology,” University of South-Eastern Norway, Autumn 2020.

[7] Wikipedia contributors, “Hydrogen Storage,” *Wikipedia, The Free Encyclopedia.*, Jan. 02, 2021. https://en.wikipedia.org/w/index.php?title=Hydrogen_storage&oldid=997829718.

[8] B. P. Xu, L. El Hima, J. X. Wen, S. Dembele, V. H. Y. Tam, and T. Donchev, “Numerical study on the spontaneous ignition of pressurized hydrogen release through a tube into air,” *Journal of Loss Prevention in the Process Industries*, vol. 21, no. 2, pp. 205–213, Mar. 2008, doi: 10.1016/j.jlp.2007.06.015.

[9] F. L. Dryer, M. Chaos, Z. Zhao, J. N. Stein, J. Y. Alpert, and C. J. Homer, “Spontaneous Ignition of Pressurized Releases of Hydrogen and Natural Gas Into Air,” *Combustion Science and Technology*, vol. 179, no. 4, pp. 663–694, Mar. 2007, doi: 10.1080/00102200600713583.

[10] Wikipedia contributors, “FLACS,” *Wikipedia, The Free Encyclopedia.*, Sep. 29, 2019. https://en.wikipedia.org/w/index.php?title=FLACS&oldid=918524572.

[11] G. Vislie, “Just How Dangerous is Hydrogen?,” presented at the Gexcon Webinar, Norway, May 15, 2020, [Online]. Available: https://www.gexcon.com/event/webinar-just-how-dangerous-is-hydrogen-promoting-hydrogen-safety/.

[12] Y. Liu, N. Tsuboi, H. Sato, F. Higashino, and A. K. Hayashi, “Direct Numerical Simulation on Hydrogen Fuel Jetting from High Pressure Tank,” Montreal, Canada, Aug. 2005.

[13] B. Maxwell and M. Radulescu, “The Spontaneous Ignition of High Pressure Hydrogen During an Accidental Release Into Air,” Montreal, Canada, May 2009, [Online]. Available: https://www.researchgate.net/publication/317503317_The_spontaneous_ignition_of_high_pressure_hydrogen_during_an_accidental_release_into_air.

[14] M. V. Bragin and V. V. Molkov, “Physics of Spontaneous Ignition of High-Pressure Hydrogen Release and Transition to Jet Fire,” *International Journal of Hydrogen Energy*, vol. 36, no. 3, pp. 2589–2596, Feb. 2011, doi: 10.1016/j.ijhydene.2010.04.128.

[15] V. V. Golub, T. V. Bazhenova, M. V. Bragin, M. F. Ivanov, and V. V. Volodin, “Hydrogen Auto-Ignition During Accidental or Technical Opening of High Pressure Tank,” presented at the 6th International Symposium on Hazards, Prevention and Mitigation of Industrial Explosions, Halifax, Canada, Sep. 2006.

[16] V. V. Golub *et al.*, “Shock-induced ignition of hydrogen gas during accidental or technical opening of high-pressure tanks,” *Journal of Loss Prevention in the Process Industries*, vol. 20, no. 4, pp. 439–446, Jul. 2007, doi: 10.1016/j.jlp.2007.03.014.

[17] V. V. Golub *et al.*, “Mechanisms of high-pressure hydrogen gas self-ignition in tubes,” *Journal of Loss Prevention in the Process Industries*, vol. 21, no. 2, pp. 185–198, Mar. 2008, doi: 10.1016/j.jlp.2007.06.012.

[18] T. V. Bazhenova, M. V. Bragin, V. V. Golub, and M. F. Ivanov, “Self-Ignition of a Fuel Gas upon Pulsed Efflux into an Oxidative Medium,” *Technical Physics Letters*, vol. 32, no. 3, pp. 269–271, 2006, doi: 10.1134/S106378500603028X.

[19] M. Li *et al.*, “Review on the research of hydrogen storage system fast refueling in fuel cell vehicle,” *International Journal of Hydrogen Energy*, vol. 44, Mar. 2019, doi: 10.1016/j.ijhydene.2019.02.208.

[20] Q. Wang, “MIT 2.097/6.339/16.920 Numerical Methods for Partial Differential Equations,” Massachusetts Institute of Technology, Spring 2015, [Online]. Available: https://www.youtube.com/c/QiqiWangGG/videos?view=0&sort=da&flow=grid.

[21] E. F.Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics*, 3rd ed. Italy: Springer, 2009.

[22] K. Vågsæther, “Modelling of Gas Explosions,” Doctoral Dissertation, Norwegian University of Science and Technology, Norway, 2010.

[23] Wikipedia contributors, “Total Variation Diminishing,” *Wikipedia, The Free Encyclopedia.*, Nov. 05, 2020. https://en.wikipedia.org/w/index.php?title=Total_variation_diminishing&oldid=987149205.

[24] Wikipedia contributors, “MUSCL scheme,” *Wikipedia, The Free Encyclopedia.*, Mar. 02, 2020. https://en.wikipedia.org/w/index.php?title=MUSCL_scheme&oldid=943585200.

[25] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics (The Finite Volume Method)*, 2nd ed. Pearson Education Limited, 2007.

[26] D. Bjerketvedt, “PEF2106-1 19V Combustion and Process Safety,” University of South-Eastern Norway, Spring 2019.

[27] R. D. McCarty, J. Hord, and H. M. Roder, “Selected Properties of Hydrogen (Engineering Design Data).” U.S. Department of Commerce/National Bureau of Standards, Feb. 1981, [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/MONO/nbsmonograph168.pdf.

[28] M. Klell, “Storage of Hydrogen in the Pure Form,” in *Handbook of Hydrogen Storage: New Materials for Future Energy Storage*, 2010, pp. 1–37.

[29] K. Groth, E. Hecht, J. T. Reynolds, M. L. Blaylock, and E. E. Carrier, “Methodology For Assessing the Safety of Hydrogen Systems: HyRAM 1.1 technical reference manual,” SAND2017-2998, 1365456, Mar. 2017. doi: 10.2172/1365456.

[30] McGough, Duncan. RhoReactingCentralFoam. C++, 2020. https://github.com/duncanam/rhoReactingCentralFoam.

[31] UniCFD. Unicfdlab/HybridCentralSolvers. C++, 2019. https://github.com/unicfdlab/hybridCentralSolvers.

[32] PisoCentralFoam. C++, 2017. https://github.com/mkraposhin/pisoCentralFoam.

[33] Romanelli, G., and E. Serioli. AeroFoam (version 2). C++, 2013. https://home.aero.polimi.it/freecase/?OpenFOAM_%2B_Code_Aster:Download.

[34] Casseau, Vincent. HyStrath. C++, 2021. https://github.com/vincentcasseau/hyStrath.

[35] M. Winter, “Benchmark and Validation of Open Source CFD Codes, With Focus on Compressible and Rotating Capabilities, For Integration on the SimScale Platform,” Master’s Thesis in Engineering Mathematics & Computational Sciences, Chalmers University of Technology, Gothenburg, Sweden, 2013.

[36] C. Greenshields, “User Guide Version 7.” The OpenFOAM Foundation, Jul. 10, 2019, [Online]. Available: https://openfoam.org.

[37] I. Choquet, “ThermophysicalModels Library in OpenFOAM-2.3.x (or 2.4.x) / How to Implement a New Thermophysical Model,” presented at the Teaching within: CFD with OpenSource software (TME050), Chalmers University of Technology, Sep. 28, 2015.

[38] D. A. McGough, “Detonation Modeling in OpenFOAM Using Adaptive Mesh Refinement,” Master’s Thesis in Aerospace Engineering, University of Colorado Boulder, Boulder, Colorado, USA, 2020.

[39] J. M. Smith, H. C. Van Ness, M. M. Abbott, and M. T. Swihart, *Introduction to Chemical Engineering Thermodynamics*, 8th ed. New York, USA: McGraw-Hill Education.

[40] Shock and Detonation Toolbox (version 2018). Python. California Institute of Technology, 2019. https://shepherd.caltech.edu/EDL/PublicResources/sdt/.

[41] Mjaavatten, Are. Rigorous Thermodynamic Models for H2O, H2, CO2, and Air (version 2.1). MATLAB, 2020. https://github.com/are-mj/thermodynamics.

[42] J. D. Andersson, *Modern Compressible Flow with Historical Perspective*, 4th ed. McGraw-Hill, 2021.

[43] B. J. McBride, S. Gordon, and M. A. Reno, “Coefficients for Calculating Thermodynamic and Transport Properties of Individual Species.” National Aeronautics and Space Administration Office of Management, 1993.

[44] B. J. McBride, M. J. Zehe, and S. Gordon, “NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species.” National Aeronautics and Space Administration / Glenn Research Center, Sep. 2002.

[45] M. W. Chase, J. L. Curnutt, J. R. Downey, R. A. McDonald, A. N. Syverud, and E. A. Valenzuela, “JANAF Thermochemical Tables, 1982 Supplement,” *Journal of Physical and Chemical Reference Data*, Oct. 2009, doi: 10.1063/1.555666.

[46] T. Allison, “JANAF Thermochemical Tables, NIST Standard Reference Database 13.” National Institute of Standards and Technology, 1998, doi: 10.18434/T42S31.

[47] J. W. Leachman, R. T. Jacobsen, S. G. Penoncello, and E. W. Lemmon, “Fundamental Equations of State for Parahydrogen, Normal Hydrogen, and Orthohydrogen,” *Journal of Physical and Chemical Reference Data*, vol. 38, no. 3, pp. 721–748, Sep. 2009, doi: 10.1063/1.3160306.

[48] H. W. Woolley, “Thermodynamic Properties of Molecular Oxygen.” U.S. Department of Commerce/National Bureau of Standards, Jun. 30, 1953.

[49] H. W. Woolley, “Thermodynamic Functions for Molecular Oxygen in the Ideal Gas State,” *Journal of Research of the National Bureau of Standards*, Feb. 1948, doi: 10.1016/0378-3812(85)87016-3.

[50] R. Span, E. W. Lemmon, R. T. Jacobsen, W. Wagner, and A. Yokozeki, “A Reference Equation of State for the Thermodynamic Properties of Nitrogen for Temperatures from 63.151 to 1000 K and Pressures to 2200 MPa,” *Journal of Physical and Chemical Reference Data*, vol. 29, no. 6, May 2001, doi: 10.1063/1.1349047.

[51] B. Lewis and G. von Elbe, *Combustion, Flames and Explosions of Gases*, 3rd ed. 1987.

[52] S. Hawksworth and L. de Broisia, “Biennal Report on Hydrogen Safety, Chapter 3,” International Association for Hydrogen Safety (HySafe), 2007. [Online]. Available: http://www.hysafe.org/documents?kwd=BRHS.

[53] S. McAllister, J.-Y. Chen, and A. C. Fernandez-Pello, *Fundamentals of Combustion Processes*. Springer.

[54] A. Kumamoto, H. Iseki, R. Ono, and T. Oda, “Measurement of Minimum Ignition Energy in Hydrogen-Oxygen-Nitrogen Premixed Gas By Spark Discharge,” presented at the 13th International Conference on Electrostatics, 2011, doi: 10.1088/1742-6596/301/1/012039.

8 Appendices

Appendix A Hydrogen Spontaneous and Pilot Ignition Limits

A.1 Hydrogen Spontaneous Ignition Limits

First of all, the hydrogen spontaneous ignition limits often called explosion limits in a homogeneous stoichiometric hydrogen-oxygen mixture should theoretically be examined. If a spherical vessel containing this mixture take into account, as seen in *Figure A.1*, there are three limits to the explosion of this mixture. The first limit is related to the wall effect; it is known that the production of radical molecules is proportional to the volume of the vessel and the destruction of them is related to the surface of the vessel; as the pressure goes down in this region, the effect of the surface becomes more and more important and for this reason, the destruction of the radicals becomes dominant and therefore more temperature will be needed to run away [26], [51]–[53].

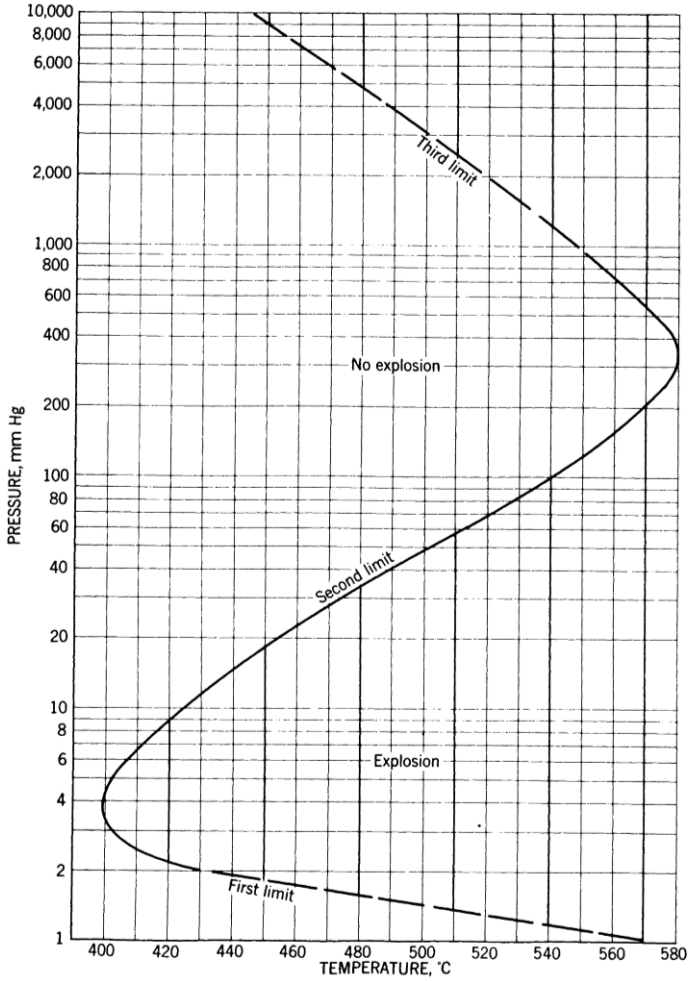


Figure A.1: Explosion limits of a stoichiometric hydrogen-oxygen mixture in a spherical KCl-coated vessel of 7.4 cm diameter [51]

The second limit is related to the balance of chain branching and chain termination reactions; in this section, the *(R A.1)* and *(R A.2)* reactions compete with each other:

$$\begin{cases} H + O_2 + M \rightarrow HO_2 + M & (Chain\ Termination) \\ H + O_2 \rightarrow OH + H & (Chain\ Branching) \end{cases} \quad \begin{matrix} \text{(R A.1)} \\ \text{(R A.2)} \end{matrix}$$

As the pressure goes up, the chain termination reaction *(R A.1)* becomes the dominant reaction, and the hydrogen radicals are consumed to produce slow hydrogen peroxide radicals (HO_2). And that is why the tendency of explosion is decreasing with increasing the pressure in this region. Finally, the third limit is related to the chain branching reactions in which the hydrogen peroxide radicals (HO_2), first convert to the hydrogen peroxide molecules (H_2O_2) and then for each molecule, two hydroxide radicals (OH) will be produced and for this reason the explosion in this region is called chain branching explosion [26], [51]–[53].

All in all, to report the critical temperature of hydrogen spontaneous ignition at atmospheric pressure, because of the safety considerations, the most violently reacting mixture, which is the stoichiometric one, in a very large volume where the wall effects can be neglected is taken into account. Based on these considerations, the critical temperature is typically reported to be 585℃ [52].

A.2 Hydrogen Minimum Ignition Energy (MIE)

As indicated in *Figure A.2*, the minimum ignition energy of hydrogen is highly affected by the hydrogen concentration in the mixture and the optimal combustion condition is at around 29% hydrogen concentration*:*

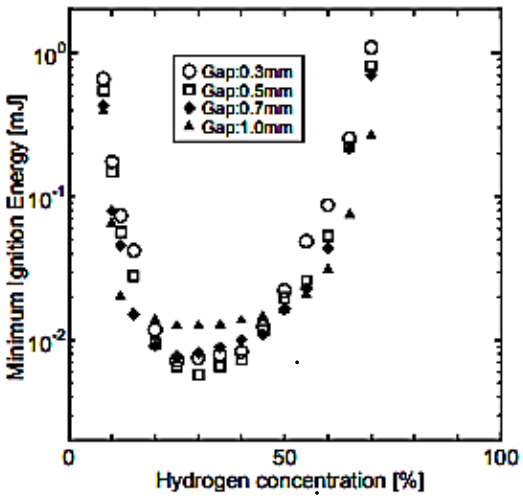


Figure A.2: MIE of $H_2 - O_2 - N_2$ mixture with fixed ratio of $O_2 / (O_2 + N_2) = 0.35$ for various H_2 and O_2 concentrations and four different gap distances [54]

A.3 Hydrogen Flammability limits

Hydrogen flammability limits will be shifted by changing the pressure and temperature of the hydrogen-air mixture, although as expected based on the logarithmic scale of the pressure axis and the linear scale of the temperature axis of *Figure A.1*, the effects of changing the temperature are much stronger than the pressure changes. The tabulated data below confirms this statement.

Table A.1: Temperature effects on hydrogen flammability limits at atmospheric pressure [52]

Temperature [℃]	Lower Flammability Limit [% volume]	Higher Flammability Limit [% volume]	Limiting Air Concentration [% volume]	Limiting Oxygen Concentration [% volume]
20	4.1	75.6	20.4	4.3
100	3.4	77.6	19.1	4.0
200	2.9	81.3	15.0	3.2
300	2.0	83.9	10.9	2.3
400	1.4	87.6	6.2	1.3

Table A.2: Pressure effects on hydrogen flammability limits at 20℃ [52]

Pressure [bar]	Lower Flammability Limit [% volume]	Higher Flammability Limit [% volume]	Limiting Air Concentration [% volume]	Limiting Oxygen Concentration [% volume]
1	4.3	78.5	21.5	4.5
10	4.9	72.4	27.6	5.8
100	5.8	74.1	25.9	5.4

In the below diagrams the effects of changing the concentration of reactants ($H_2, O_2, and\ N_2$) in two different temperatures, 20℃ and 400℃, and atmospheric pressure on the flammability limits are shown.

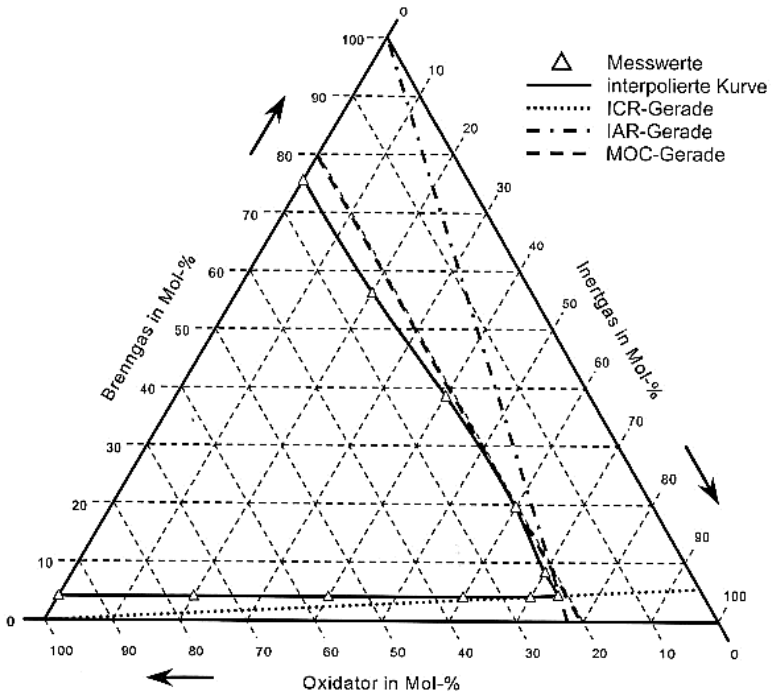


Figure A.3: Flammability diagram for $H_2 - O_2 - N_2$ mixture at 20℃ and atmospheric pressure [52]

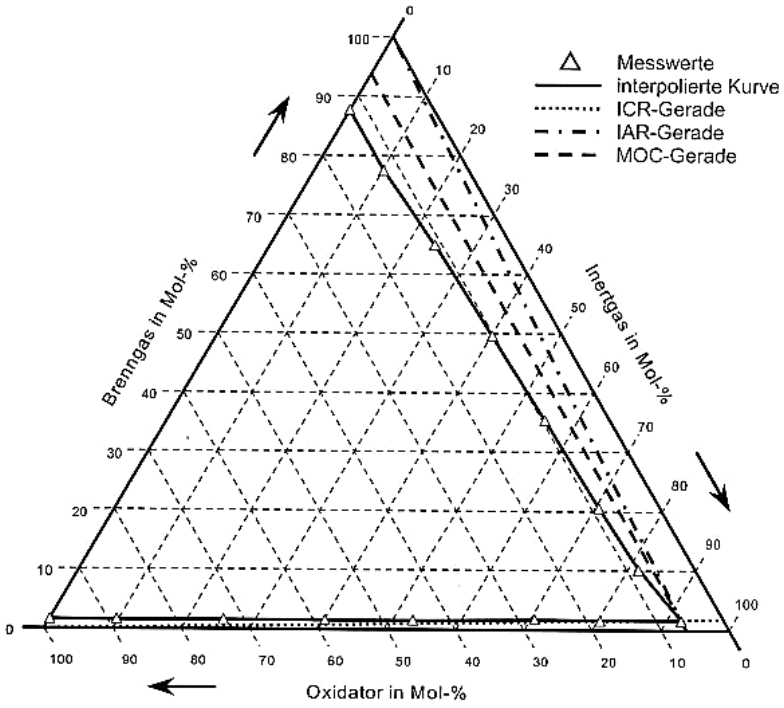


Figure A.4: Flammability diagram for $H_2 - O_2 - N_2$ mixture at 400℃ and atmoshpheric pressure [52]

Appendix B Hydrogen Storage Approaches

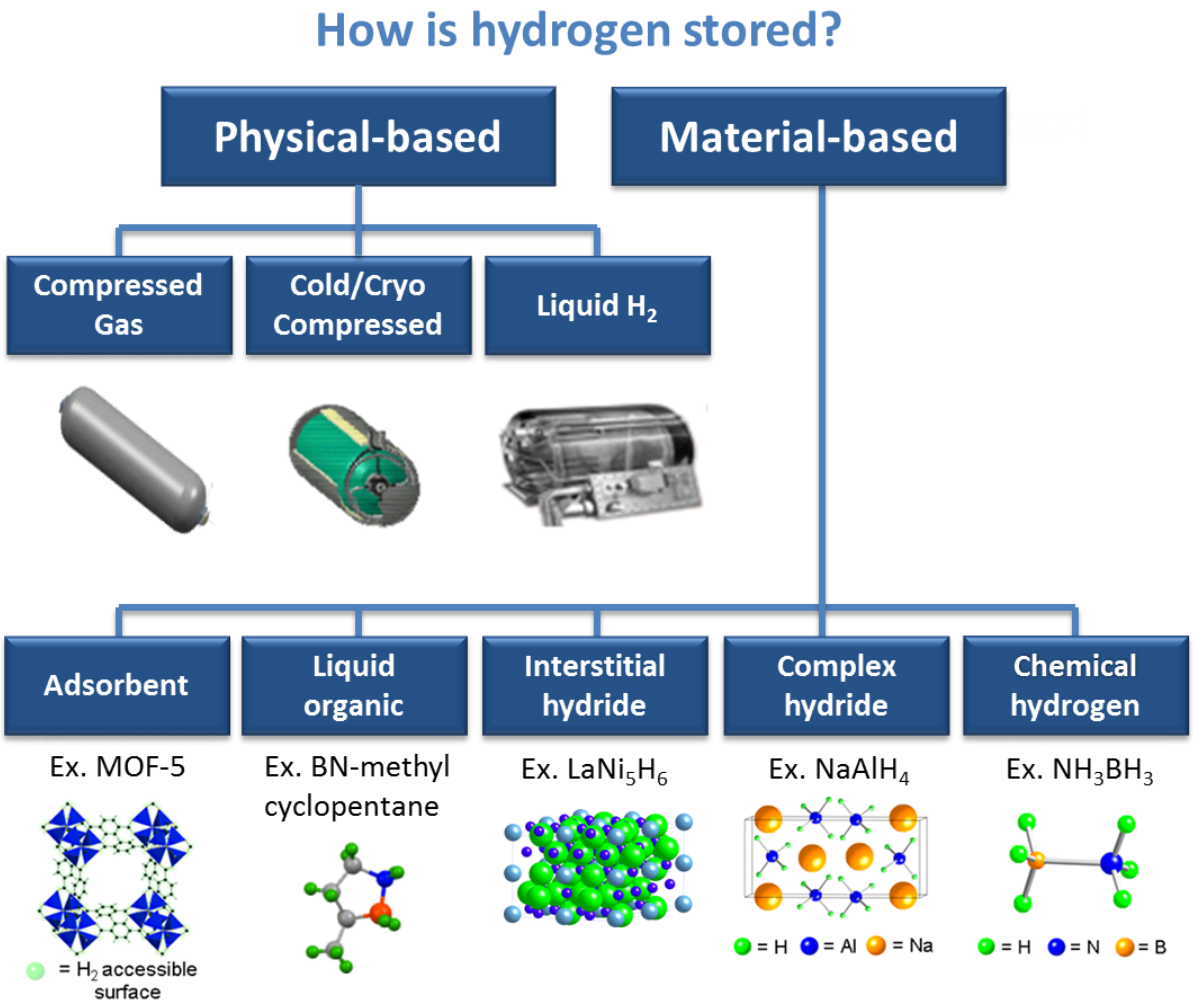


Figure B.1: Subcategories of the two main hydrogen storage approaches, physical-based storage and material-based storage [5]

Appendix C The Algorithm (Flowchart) of the Original USN-FLIC Code

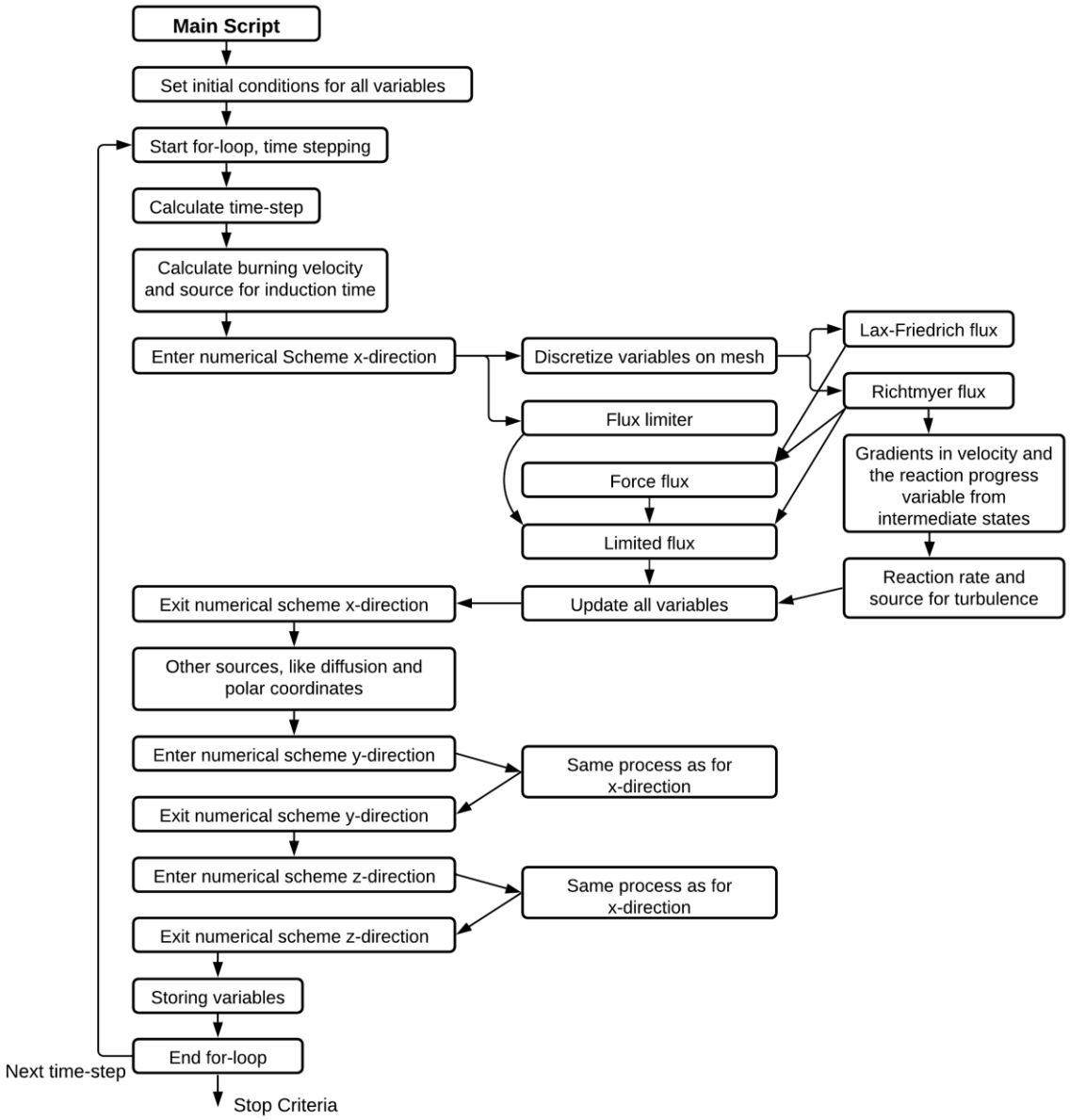


Figure C.1: Flowchart of the original USN-FLIC code [22]

Appendix D The Core Section of the Particle Tracking function

This function is written based on the variables of the USN-FLIC code and in order to run it needs (I) the initial position of the specific particle (`X_p_0`, `Y_p_0`), (II) the time when the particle was at this point (`t_p_0`), (III) the real time of the simulation (`time`), (IV) and the stored velocity matrices in x and y directions (`ux` and `uy`) corresponding to this time, (V) and finally, the grid resolutions (`dx`, `dy`).

```
function [X_p,Y_p,time_p] = pathLine(X_p_0,Y_p_0,time,time_0,t_p_0,ux,uy,j,dx,dy)

%%%%%%%%%%%%%%%%%%%%%%%%%%% Particle Tracking %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function is written to find the pathline (particle tracking) of a  %
% specified particle with entering the initial location(X0_p,Y0_p) of that%
% particle in the domain.                                                 %
% Keivan A.Gh   May,2020                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fileName_ux_0 = ['C:\Users\...\ux_',num2str(j-0.5),'.mat'];
fileName_uy_0 = ['C:\Users\...\uy_',num2str(j-0.5),'.mat'];

ux_0 = importdata(fileName_ux_0);
uy_0 = importdata(fileName_uy_0);

dt_p = time - t_p_0;
ii = fix(X_p_0);
jj = fix(Y_p_0);

%%
ux_ave = (ux_0(ii,jj)+ux_0(ii+1,jj)+ux_0(ii,jj+1)+ux_0(ii+1,jj+1))/4;
uy_ave = (uy_0(ii,jj)+uy_0(ii+1,jj)+uy_0(ii,jj+1)+uy_0(ii+1,jj+1))/4;

X_p = (ux_ave * dt_p)/dx + X_p_0;
Y_p = (uy_ave * dt_p)/dy + Y_p_0;

if X_p >= ii+1
    dt_x = ((ii+1) - X_p_0)*dx/ux_ave;
elseif X_p < ii
    dt_x = (ii - X_p_0)*dx/ux_ave;
else
    dt_x = dt_p;
end

if Y_p >= jj+1
    dt_y = ((jj+1) - Y_p_0)*dy/uy_ave;
elseif Y_p < jj
    dt_y = (jj - Y_p_0)*dy/uy_ave;
else
    dt_y = dt_p;
end

if dt_x == 0 && dt_y == 0
    ux_ave_0 = (ux_0(ii-1,jj-1)+ux_0(ii,jj-1)+ux_0(ii-1,jj)+ux_0(ii,jj))/4;
    uy_ave_0 = (uy_0(ii-1,jj-1)+uy_0(ii,jj-1)+uy_0(ii-1,jj)+uy_0(ii,jj))/4;
    if ux_ave_0 >= 0 && uy_ave_0 >=0

        dt_x = dt_p;
        dt_y = dt_p;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = X_p_0;
        Y_p = Y_p_0;
```

```
    elseif ux_ave_0 >= 0 && uy_ave_0 < 0

        dt_x = dt_p;
        dt_y = -dy/uy_ave_0;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = X_p_0;
        Y_p = (uy_ave_0 * dt_p)/dy + Y_p_0;

    elseif ux_ave_0 < 0 && uy_ave_0 >= 0

        dt_x = -dx/ux_ave_0;
        dt_y = dt_p;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = (ux_ave_0 * dt_p)/dx + X_p_0;
        Y_p = Y_p_0;
    else
        dt_x = -dx/ux_ave_0;
        dt_y = -dy/uy_ave_0;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = (ux_ave_0 * dt_p)/dx + X_p_0;
        Y_p = (uy_ave_0 * dt_p)/dy + Y_p_0;

    end

elseif dt_x == 0 && dt_y ~= 0
    ux_ave_0 = (ux_0(ii-1,jj)+ux_0(ii,jj)+ux_0(ii-1,jj+1)+ux_0(ii,jj+1))/4;
    uy_ave_0 = (uy_0(ii-1,jj)+uy_0(ii,jj)+uy_0(ii-1,jj+1)+uy_0(ii,jj+1))/4;

    Y_p = (uy_ave_0 * dt_p)/dy + Y_p_0;
    if Y_p >= jj+1
        dt_y = ((jj+1) - Y_p_0)*dy/uy_ave_0;
    elseif Y_p < jj
        dt_y = (jj - Y_p_0)*dy/uy_ave_0;
    else
        dt_y = dt_p;
    end

    if ux_ave_0 >= 0

        dt_x = dt_p;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = X_p_0;
        Y_p = (uy_ave_0 * dt_p)/dy + Y_p_0;
    else
        dt_x = -dx/ux_ave_0;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = (ux_ave_0 * dt_p)/dx + X_p_0;
        Y_p = (uy_ave_0 * dt_p)/dy + Y_p_0;
    end

elseif dt_x ~= 0 && dt_y == 0
    ux_ave_0 = (ux_0(ii,jj-1)+ux_0(ii+1,jj-1)+ux_0(ii,jj)+ux_0(ii+1,jj))/4;
    uy_ave_0 = (uy_0(ii,jj-1)+uy_0(ii+1,jj-1)+uy_0(ii,jj)+uy_0(ii+1,jj))/4;

    X_p = (ux_ave_0 * dt_p)/dx + X_p_0;
    if X_p >= ii+1
        dt_x = ((ii+1) - X_p_0)*dx/ux_ave_0;
    elseif X_p < ii
        dt_x = (ii - X_p_0)*dx/ux_ave_0;
    else
```

```
        dt_x = dt_p;
    end

    if uy_ave_0 >= 0

        dt_y = dt_p;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = (ux_ave_0 * dt_p)/dx + X_p_0;
        Y_p = Y_p_0;
    else
        dt_y = -dy/uy_ave_0;
        dt_p = min([dt_x,dt_y,dt_p]);

        X_p = (ux_ave_0 * dt_p)/dx + X_p_0;
        Y_p = (uy_ave_0 * dt_p)/dy + Y_p_0;
    end
else
    dt_p = min([dt_x,dt_y,dt_p]);

    X_p = (ux_ave * dt_p)/dx + X_p_0;
    Y_p = (uy_ave * dt_p)/dy + Y_p_0;
end

time_p = t_p_0 + dt_p;
%[X_p Y_p time_p]
```

Appendix E Settings of the System Sub-Dictionaries

E.1 controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     rhoReactingCentralFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         1e-04;

deltaT          1e-09;

writeControl    adjustableRunTime;

writeInterval   5e-07;

purgeWrite      0;

writeFormat     binary;

writePrecision  8;

writeCompression off;

timeFormat      general;

timePrecision   6;

graphFormat     raw;

runTimeModifiable   true;

adjustTimeStep      yes;

useAcousticCourant  yes;

maxCo               0.1;

maxDeltaT           1e-06;

maxAcousticCo       0.1;

functions
{
       #includeFunc residuals

       #includeFunc minMaxMagnitude
}


// ************************************************************************* //
```

E.2 fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

fluxScheme      Kurganov;

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(tauMC)      Gauss linear;
    div(phi,O2)     Gauss vanLeer01;
    div(phi,H2)     Gauss vanLeer01;
    div(phi,H2O)    Gauss vanLeer01;
    div(phi,N2)     Gauss vanLeer01;
    div(phi,Yi_h)   Gauss vanLeer01;

    div(phi,k)      Gauss vanLeer;
    div(phi,omega)  Gauss vanLeer;
}

laplacianSchemes
{
    default         Gauss linear uncorrected;
}

interpolationSchemes
{
    default         linear;
    reconstruct(rho)vanLeer;
    reconstruct(U)  vanLeerV;
    reconstruct(T)  vanLeer;
    reconstruct(Yi) vanLeer;
}

snGradSchemes
{
    default         uncorrected;
}

wallDist
{
    method meshWave;
}

// ************************************************************************* //
```

E.3 fvSolutions

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    "(rho|rhoU|rhoE)"
    {
        solver          diagonal;
    }
    U
    {
        solver          PBiCGStab;
        preconditioner  DIC;
        tolerance       1e-15;
        relTol          0;
    }
    "(k|epsilon|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-06;
        relTol          0.01;
    }
    "(k|epsilon|omega)Final"
    {
        $U;
        tolerance       1e-06;
        relTol          0;
    }
    h
    {
        $U;
        tolerance       1e-10;
        relTol          0;
    }
    e
    {
        $U;
        tolerance       1e-10;
        relTol          0;
    }
    "(O2|H2|H2O|N2|Yi)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-10;
        relTol          0;
    }
}
```

```
CENTRAL
{
}

relaxationFactors
{
    equations
    {
        ".*"            0.3;
    }
}

// ************************************************************************* //
```

Appendix F thermo.compressibleGas Dictionary for the Initial Simulations

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermo.compressibleGas;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
O2
{
    specie
    {
        molWeight       31.9988;
    }
    thermodynamics
    {
        Tlow            200;
        Thigh           5000;
        Tcommon         1000;
        highCpCoeffs    ( 3.28253784 0.00148308754 -7.57966669e-07 2.09470555e-10
                          -2.16717794e-14 -1088.45772 5.45323129 );
        lowCpCoeffs     ( 3.78245636 -0.00299673416 9.84730201e-06 -9.68129509e-09
                          3.24372837e-12 -1063.94356 3.65767573 );
    }
    transport
    {
        As              1.512e-06;
        Ts              120;
    }
    elements
    {
        O               2;
    }
    equationOfState
    {
        Tc              154.6;
        Vc              73.4e-03;
        Zc              0.288;
        Pc              50.43e05;
        omega           0.022;
    }
}

H2
{
    specie
    {
        molWeight       2.01594;
    }
    thermodynamics
    {
        Tlow            200;
        Thigh           5000;
        Tcommon         1000;
```

```
        highCpCoeffs    ( 3.3372792 -4.94024731e-05 4.99456778e-07 -1.79566394e-10
                          2.00255376e-14 -950.158922 -3.20502331 );
        lowCpCoeffs     ( 2.34433112 0.00798052075 -1.9478151e-05 2.01572094e-08
                          -7.37611761e-12 -917.935173 0.683010238 );
    }
    transport
    {
        As              6.362e-07;
        Ts              72;
    }
    elements
    {
        H               2;
    }
    equationOfState
    {
        Tc              33.19;
        Vc              64.1e-03;
        Zc              0.305;
        Pc              13.13e05;
        omega           -0.216;
    }
}

N2
{
    specie
    {
        molWeight       28.0134;
    }
    thermodynamics
    {
        Tlow            250;
        Thigh           5000;
        Tcommon         1000;
        highCpCoeffs    ( 2.92664 0.0014879768 -5.68476e-07 1.0097038e-10
                          -6.753351e-15 -922.7977 5.980528 );
        lowCpCoeffs     ( 3.298677 0.0014082404 -3.963222e-06 5.641515e-09
                          -2.444854e-12 -1020.8999 3.950372 );
    }
    transport
    {
        As              1.512e-06;
        Ts              120;
    }
    elements
    {
        N               2;
    }
    equationOfState
    {
        Tc              126.2;
        Vc              89.2e-03;
        Zc              0.289;
        Pc              34.00e05;
        omega           0.038;
    }
}

// ************************************************************************* //
```

Appendix G Implemented nasa9Poly Thermodynamic Model

G.1 nasa9PolyThermo.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Keivan A.Gh / 28, Oct, 2020:
Description: Tcommon1 and Tcommon2 could be the same for those elements and binary
             molecules that do not have 3 different regions! For these kind of
             data midCpCoeffs could be the same as highCpCoeffs.
\*---------------------------------------------------------------------------*/

#include "nasa9PolyThermo.H"
#include "IOstreams.H"

// * * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * * * //

template<class EquationOfState>
void Foam::nasa9PolyThermo<EquationOfState>::checkInputData() const
{
    if (Tlow_ >= Thigh_)
    {
        FatalErrorInFunction
            << "Tlow(" << Tlow_ << ") >= Thigh(" << Thigh_ << ')'
            << exit(FatalError);
    }

    if (Tcommon1_ <= Tlow_)
    {
        FatalErrorInFunction
            << "Tcommon1(" << Tcommon1_ << ") <= Tlow(" << Tlow_ << ')'
            << exit(FatalError);
    }

     if (Tcommon2_ < Tcommon1_)
    {
        FatalErrorInFunction
            << "Tcommon2(" << Tcommon2_ << ") < Tcommon1(" << Tcommon1_ << ')'
            << exit(FatalError);
    }

    if (Tcommon2_ > Thigh_)
    {
        FatalErrorInFunction
            << "Tcommon2(" << Tcommon2_ << ") > Thigh(" << Thigh_ << ')'
            << exit(FatalError);
    }
}
```

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

template<class EquationOfState>
Foam::nasa9PolyThermo<EquationOfState>::nasa9PolyThermo(const dictionary& dict)
:
    EquationOfState(dict),
    Tlow_(readScalar(dict.subDict("thermodynamics").lookup("Tlow"))),
    Thigh_(readScalar(dict.subDict("thermodynamics").lookup("Thigh"))),
    Tcommon1_(readScalar(dict.subDict("thermodynamics").lookup("Tcommon1"))),
    Tcommon2_(readScalar(dict.subDict("thermodynamics").lookup("Tcommon2"))),
    highCpCoeffs_(dict.subDict("thermodynamics").lookup("highCpCoeffs")),
    midCpCoeffs_(dict.subDict("thermodynamics").lookup("midCpCoeffs")),
    lowCpCoeffs_(dict.subDict("thermodynamics").lookup("lowCpCoeffs"))
{
    // Convert coefficients to mass-basis
    for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
    {
        highCpCoeffs_[coefLabel] *= this->R();
        midCpCoeffs_[coefLabel] *= this->R();
        lowCpCoeffs_[coefLabel] *= this->R();
    }

    checkInputData();
}
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * //

template<class EquationOfState>
void Foam::nasa9PolyThermo<EquationOfState>::write(Ostream& os) const
{
    EquationOfState::write(os);

    // Convert coefficients back to dimensionless form
    coeffArray highCpCoeffs;
    coeffArray midCpCoeffs;
    coeffArray lowCpCoeffs;
    for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
    {
        highCpCoeffs[coefLabel] = highCpCoeffs_[coefLabel]/this->R();
        midCpCoeffs[coefLabel] = midCpCoeffs_[coefLabel]/this->R();
        lowCpCoeffs[coefLabel] = lowCpCoeffs_[coefLabel]/this->R();
    }

    dictionary dict("thermodynamics");
    dict.add("Tlow", Tlow_);
    dict.add("Thigh", Thigh_);
    dict.add("Tcommon1", Tcommon1_);
    dict.add("Tcommon2", Tcommon2_);
    dict.add("highCpCoeffs", highCpCoeffs);
    dict.add("midCpCoeffs", midCpCoeffs);
    dict.add("lowCpCoeffs", lowCpCoeffs);
    os  << indent << dict.dictName() << dict;
}
// * * * * * * * * * * * * * * * * * * Ostream Operator  * * * * * * * * * * * * * * * * * //

template<class EquationOfState>
Foam::Ostream& Foam::operator<<
(
    Ostream& os,
    const nasa9PolyThermo<EquationOfState>& jt
)
{
    jt.write(os);
    return os;
}

// ************************************************************************* //
```

G.2 nasa9PolyThermo.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2019 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Keivan A.Gh / 28, Oct, 2020:
Class
    Foam::nasa9PolyThermo

Description
    JANAF tables based thermodynamics package (with 9 Coefficients) templated
    into the equation of state.

SourceFiles
    nasa9PolyThermoI.H
    nasa9PolyThermo.C

\*---------------------------------------------------------------------------*/

#ifndef nasa9PolyThermo_H
#define nasa9PolyThermo_H

#include "scalar.H"
#include "FixedList.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

// Forward declaration of friend functions and operators

template<class EquationOfState> class nasa9PolyThermo;

template<class EquationOfState>
inline nasa9PolyThermo<EquationOfState> operator+
(
    const nasa9PolyThermo<EquationOfState>&,
    const nasa9PolyThermo<EquationOfState>&
);

template<class EquationOfState>
inline nasa9PolyThermo<EquationOfState> operator*
(
    const scalar,
    const nasa9PolyThermo<EquationOfState>&
);

template<class EquationOfState>
inline nasa9PolyThermo<EquationOfState> operator==
(
    const nasa9PolyThermo<EquationOfState>&,
```

```
    const nasa9PolyThermo<EquationOfState>&
);

template<class EquationOfState>
Ostream& operator<<
(
    Ostream&,
    const nasa9PolyThermo<EquationOfState>&
);

/*---------------------------------------------------------------------------*\
                           Class nasa9PolyThermo Declaration
\*---------------------------------------------------------------------------*/

template<class EquationOfState>
class nasa9PolyThermo
:
    public EquationOfState
{

public:

    // Public data

        static const int nCoeffs_ = 9;
        typedef FixedList<scalar, nCoeffs_> coeffArray;


private:

    // Private Data

        // Temperature limits of applicability of functions
        scalar Tlow_, Thigh_, Tcommon1_, Tcommon2_;

        coeffArray highCpCoeffs_;
        coeffArray midCpCoeffs_;
        coeffArray lowCpCoeffs_;

    // Private Member Functions

        //- Check that input data is valid
        void checkInputData() const;

        //- Return the coefficients corresponding to the given temperature
        inline const coeffArray& coeffs(const scalar T) const;

public:

    // Constructors

        //- Construct from components
        inline nasa9PolyThermo
        (
            const EquationOfState& st,
            const scalar Tlow,
            const scalar Thigh,
            const scalar Tcommon1,
            const scalar Tcommon2,
            const coeffArray& highCpCoeffs,
            const coeffArray& midCpCoeffs,
            const coeffArray& lowCpCoeffs,
            const bool convertCoeffs = false
        );

        //- Construct from dictionary
        nasa9PolyThermo(const dictionary& dict);

        //- Construct as a named copy
        inline nasa9PolyThermo(const word&, const nasa9PolyThermo&);

    // Member Functions

        //- Return the instantiated type name
```

```
    static word typeName()
    {
        return "nasa9Poly<" + EquationOfState::typeName() + '>';
    }

    //- Limit the temperature to be in the range Tlow_ to Thigh_
    inline scalar limit(const scalar T) const;

    // Access

        //- Return const access to the low temperature limit
        inline scalar Tlow() const;

        //- Return const access to the high temperature limit
        inline scalar Thigh() const;

        //- Return const access to the common1 temperature
        inline scalar Tcommon1() const;

         //- Return const access to the common2 temperature
        inline scalar Tcommon2() const;

        //- Return const access to the high temperature poly coefficients
        inline const coeffArray& highCpCoeffs() const;

         //- Return const access to the middle temperature poly coefficients
        inline const coeffArray& midCpCoeffs() const;

        //- Return const access to the low temperature poly coefficients
        inline const coeffArray& lowCpCoeffs() const;

    // Fundamental properties

        //- Heat capacity at constant pressure [J/kg/K]
        inline scalar Cp(const scalar p, const scalar T) const;

        //- Absolute Enthalpy [J/kg]
        inline scalar Ha(const scalar p, const scalar T) const;

        //- Sensible enthalpy [J/kg]
        inline scalar Hs(const scalar p, const scalar T) const;

        //- Chemical enthalpy [J/kg]
        inline scalar Hc() const;

        //- Entropy [J/kg/K]
        inline scalar S(const scalar p, const scalar T) const;

        #include "HtoEthermo.H"

    // Derivative term used for Jacobian

        //- Derivative of Gibbs free energy w.r.t. temperature
        inline scalar dGdT(const scalar p, const scalar T) const;

        //- Temperature derivative of heat capacity at constant pressure
        inline scalar dCpdT(const scalar p, const scalar T) const;

    // I-O

        //- Write to Ostream
        void write(Ostream& os) const;

// Member Operators

    inline void operator+=(const nasa9PolyThermo&);

// Friend operators

    friend nasa9PolyThermo operator+ <EquationOfState>
    (
        const nasa9PolyThermo&,
        const nasa9PolyThermo&
    );
```

```
        friend nasa9PolyThermo operator* <EquationOfState>
        (
            const scalar,
            const nasa9PolyThermo&
        );

        friend nasa9PolyThermo operator== <EquationOfState>
        (
            const nasa9PolyThermo&,
            const nasa9PolyThermo&
        );

    // Ostream Operator

        friend Ostream& operator<< <EquationOfState>
        (
            Ostream&,
            const nasa9PolyThermo&
        );
};

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include "nasa9PolyThermoI.H"

#ifdef NoRepository
    #include "nasa9PolyThermo.C"
#endif

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************************************************************* //
```

G.3 nasa9PolyThermoI.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Keivan A.Gh / 28, Oct, 2020:

\*---------------------------------------------------------------------------*/

#include "nasa9PolyThermo.H"
#include "specie.H"
```

```
// * * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * * * //

template<class EquationOfState>
inline Foam::nasa9PolyThermo<EquationOfState>::nasa9PolyThermo
(
    const EquationOfState& st,
    const scalar Tlow,
    const scalar Thigh,
    const scalar Tcommon1,
    const scalar Tcommon2,
    const typename nasa9PolyThermo<EquationOfState>::coeffArray& highCpCoeffs,
    const typename nasa9PolyThermo<EquationOfState>::coeffArray& midCpCoeffs,
    const typename nasa9PolyThermo<EquationOfState>::coeffArray& lowCpCoeffs,
    const bool convertCoeffs
)
:
    EquationOfState(st),
    Tlow_(Tlow),
    Thigh_(Thigh),
    Tcommon1_(Tcommon1),
    Tcommon2_(Tcommon2)
{
    if (convertCoeffs)
    {
        for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
        {
            highCpCoeffs_[coefLabel] = highCpCoeffs[coefLabel]*this->R();
            midCpCoeffs_[coefLabel] = midCpCoeffs[coefLabel]*this->R();
            lowCpCoeffs_[coefLabel] = lowCpCoeffs[coefLabel]*this->R();
        }
    }
    else
    {
        for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
        {
            highCpCoeffs_[coefLabel] = highCpCoeffs[coefLabel];
            midCpCoeffs_[coefLabel] = midCpCoeffs[coefLabel];
            lowCpCoeffs_[coefLabel] = lowCpCoeffs[coefLabel];
        }
    }
}

template<class EquationOfState>
inline const typename Foam::nasa9PolyThermo<EquationOfState>::coeffArray&
Foam::nasa9PolyThermo<EquationOfState>::coeffs
(
    const scalar T
) const
{
    if (T < Tcommon1_)
    {
        return lowCpCoeffs_;
    }
     else if (T < Tcommon2_)
     {
        return midCpCoeffs_;
     }
    else
    {
        return highCpCoeffs_;
    }
}

// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //

template<class EquationOfState>
```

```
inline Foam::nasa9PolyThermo<EquationOfState>::nasa9PolyThermo
(
    const word& name,
    const nasa9PolyThermo& jt
)
:
    EquationOfState(name, jt),
    Tlow_(jt.Tlow_),
    Thigh_(jt.Thigh_),
    Tcommon1_(jt.Tcommon1_),
    Tcommon2_(jt.Tcommon2_)
{
    for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
    {
        highCpCoeffs_[coefLabel] = jt.highCpCoeffs_[coefLabel];
        midCpCoeffs_[coefLabel] = jt.midCpCoeffs_[coefLabel];
        lowCpCoeffs_[coefLabel] = jt.lowCpCoeffs_[coefLabel];
    }
}

// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::limit
(
    const scalar T
) const
{
    if (T < Tlow_ || T > Thigh_)
    {
        WarningInFunction
            << "attempt to use nasa9PolyThermo<EquationOfState>"
               " out of temperature range "
            << Tlow_ << " -> " << Thigh_ << ";  T = " << T
            << endl;

        return min(max(T, Tlow_), Thigh_);
    }
    else
    {
        return T;
    }
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Tlow() const
{
    return Tlow_;
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Thigh() const
{
    return Thigh_;
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Tcommon1() const
{
    return Tcommon1_;
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Tcommon2() const
{
    return Tcommon2_;
}
```

```
template<class EquationOfState>
inline const typename Foam::nasa9PolyThermo<EquationOfState>::coeffArray&
Foam::nasa9PolyThermo<EquationOfState>::highCpCoeffs() const
{
    return highCpCoeffs_;
}

template<class EquationOfState>
inline const typename Foam::nasa9PolyThermo<EquationOfState>::coeffArray&
Foam::nasa9PolyThermo<EquationOfState>::midCpCoeffs() const
{
    return midCpCoeffs_;
}

template<class EquationOfState>
inline const typename Foam::nasa9PolyThermo<EquationOfState>::coeffArray&
Foam::nasa9PolyThermo<EquationOfState>::lowCpCoeffs() const
{
    return lowCpCoeffs_;
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Cp
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return
        ((((a[6]*T + a[5])*T + a[4])*T + a[3])*T + a[2])
        + (a[0]/T + a[1])/T
        + EquationOfState::Cp(p, T);
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Ha
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return
    (
       ((((a[6]/5.0*T + a[5]/4.0)*T + a[4]/3.0)*T + a[3]/2.0)*T + a[2])*T
       + (-a[0]/T + a[1]*log(T))
       + a[7]
    )  + EquationOfState::H(p, T);
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Hs
(
    const scalar p,
    const scalar T
) const
{
    return Ha(p, T) - Hc();
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::Hc() const
{
    const coeffArray& a = lowCpCoeffs_;
    return
    (
```

```
        ((((a[6]/5.0*Tstd + a[5]/4.0)*Tstd + a[4]/3.0)*Tstd + a[3]/2.0)*Tstd
       + a[2])*Tstd
       + (-a[0]/Tstd + a[1]*log(Tstd))
       + a[7]
    );
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::S
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return
    (
       (((a[6]/4.0*T + a[5]/3.0)*T + a[4]/2.0)*T + a[3])*T + a[2]*log(T)
       + (-a[0]/2.0/T - a[1])/T
       + a[8]
    )  + EquationOfState::S(p, T);
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::dGdT
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return -(
               ((-a[0]/T + a[1]*log(T))/T + a[2] + a[7]/T)/T
               + a[3]/2 + T*(a[4]/3 + T*(a[5]/4 + T*a[6]/5))
            );
}

template<class EquationOfState>
inline Foam::scalar Foam::nasa9PolyThermo<EquationOfState>::dCpdT
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return
        (
             (((4*a[6]*T + 3*a[5])*T + 2*a[4])*T + a[3])
             + ((-2*a[0]/T - a[1])/T)/T
        );
}

// * * * * * * * * * * * * * * * * Member Operators  * * * * * * * * * * * * * //

template<class EquationOfState>
inline void Foam::nasa9PolyThermo<EquationOfState>::operator+=
(
    const nasa9PolyThermo<EquationOfState>& jt
)
{
    scalar Y1 = this->Y();

    EquationOfState::operator+=(jt);

    if (mag(this->Y()) > small)
    {
        Y1 /= this->Y();
```

```
        const scalar Y2 = jt.Y()/this->Y();

        Tlow_ = max(Tlow_, jt.Tlow_);
        Thigh_ = min(Thigh_, jt.Thigh_);

        if
        (
            nasa9PolyThermo<EquationOfState>::debug
         && notEqual(Tcommon1_, jt.Tcommon1_)
        )
        {
            FatalErrorInFunction
                << "Tcommon1 " << Tcommon1_ << " for "
                << (this->name().size() ? this->name() : "others")
                << " != " << jt.Tcommon1_ << " for "
                << (jt.name().size() ? jt.name() : "others")
                << exit(FatalError);
        }

       if
        (
            nasa9PolyThermo<EquationOfState>::debug
         && notEqual(Tcommon2_, jt.Tcommon2_)
        )
        {
            FatalErrorInFunction
                << "Tcommon2 " << Tcommon2_ << " for "
                << (this->name().size() ? this->name() : "others")
                << " != " << jt.Tcommon2_ << " for "
                << (jt.name().size() ? jt.name() : "others")
                << exit(FatalError);
        }

        for
        (
            label coefLabel=0;
            coefLabel<nasa9PolyThermo<EquationOfState>::nCoeffs_;
            coefLabel++
        )
        {
            highCpCoeffs_[coefLabel] =
                Y1*highCpCoeffs_[coefLabel]
              + Y2*jt.highCpCoeffs_[coefLabel];

             midCpCoeffs_[coefLabel] =
                Y1*midCpCoeffs_[coefLabel]
              + Y2*jt.midCpCoeffs_[coefLabel];

            lowCpCoeffs_[coefLabel] =
                Y1*lowCpCoeffs_[coefLabel]
              + Y2*jt.lowCpCoeffs_[coefLabel];
        }
    }
}

// * * * * * * * * * * * * * * * * Friend Operators  * * * * * * * * * * * * * //

template<class EquationOfState>
inline Foam::nasa9PolyThermo<EquationOfState> Foam::operator+
(
    const nasa9PolyThermo<EquationOfState>& jt1,
    const nasa9PolyThermo<EquationOfState>& jt2
)
{
    EquationOfState eofs = jt1;
    eofs += jt2;
```

```
if (mag(eofs.Y()) < small)
{
    return nasa9PolyThermo<EquationOfState>
    (
        eofs,
        jt1.Tlow_,
        jt1.Thigh_,
        jt1.Tcommon1_,
        jt1.Tcommon2_,
        jt1.highCpCoeffs_,
        jt1.midCpCoeffs_,
        jt1.lowCpCoeffs_
    );
}
else
{
    const scalar Y1 = jt1.Y()/eofs.Y();
    const scalar Y2 = jt2.Y()/eofs.Y();

    typename nasa9PolyThermo<EquationOfState>::coeffArray highCpCoeffs;
    typename nasa9PolyThermo<EquationOfState>::coeffArray midCpCoeffs;
    typename nasa9PolyThermo<EquationOfState>::coeffArray lowCpCoeffs;

    for
    (
        label coefLabel=0;
        coefLabel<nasa9PolyThermo<EquationOfState>::nCoeffs_;
        coefLabel++
    )
    {
        highCpCoeffs[coefLabel] =
            Y1*jt1.highCpCoeffs_[coefLabel]
          + Y2*jt2.highCpCoeffs_[coefLabel];

         midCpCoeffs[coefLabel] =
            Y1*jt1.midCpCoeffs_[coefLabel]
          + Y2*jt2.midCpCoeffs_[coefLabel];

        lowCpCoeffs[coefLabel] =
            Y1*jt1.lowCpCoeffs_[coefLabel]
          + Y2*jt2.lowCpCoeffs_[coefLabel];
    }

    if
    (
        nasa9PolyThermo<EquationOfState>::debug
     && notEqual(jt1.Tcommon1_, jt2.Tcommon1_)
    )
    {
        FatalErrorInFunction
            << "Tcommon1 " << jt1.Tcommon1_ << " for "
            << (jt1.name().size() ? jt1.name() : "others")
            << " != " << jt2.Tcommon1_ << " for "
            << (jt2.name().size() ? jt2.name() : "others")
            << exit(FatalError);
    }

   if
    (
        nasa9PolyThermo<EquationOfState>::debug
     && notEqual(jt1.Tcommon2_, jt2.Tcommon2_)
    )
    {
        FatalErrorInFunction
            << "Tcommon2 " << jt1.Tcommon2_ << " for "
            << (jt1.name().size() ? jt1.name() : "others")
            << " != " << jt2.Tcommon2_ << " for "
```

```
                << (jt2.name().size() ? jt2.name() : "others")
                << exit(FatalError);
        }

        return nasa9PolyThermo<EquationOfState>
        (
            eofs,
            max(jt1.Tlow_, jt2.Tlow_),
            min(jt1.Thigh_, jt2.Thigh_),
            jt1.Tcommon1_,
            jt1.Tcommon2_,
            highCpCoeffs,
            midCpCoeffs,
            lowCpCoeffs
        );
    }
}

template<class EquationOfState>
inline Foam::nasa9PolyThermo<EquationOfState> Foam::operator*
(
    const scalar s,
    const nasa9PolyThermo<EquationOfState>& jt
)
{
    return nasa9PolyThermo<EquationOfState>
    (
        s*static_cast<const EquationOfState&>(jt),
        jt.Tlow_,
        jt.Thigh_,
        jt.Tcommon1_,
        jt.Tcommon2_,
        jt.highCpCoeffs_,
        jt.midCpCoeffs_,
        jt.lowCpCoeffs_
    );
}

template<class EquationOfState>
inline Foam::nasa9PolyThermo<EquationOfState> Foam::operator==
(
    const nasa9PolyThermo<EquationOfState>& jt1,
    const nasa9PolyThermo<EquationOfState>& jt2
)
{
    EquationOfState eofs
    (
        static_cast<const EquationOfState&>(jt1)
     == static_cast<const EquationOfState&>(jt2)
    );

    const scalar Y1 = jt2.Y()/eofs.Y();
    const scalar Y2 = jt1.Y()/eofs.Y();

    typename nasa9PolyThermo<EquationOfState>::coeffArray highCpCoeffs;
    typename nasa9PolyThermo<EquationOfState>::coeffArray midCpCoeffs;
    typename nasa9PolyThermo<EquationOfState>::coeffArray lowCpCoeffs;

    for
    (
        label coefLabel=0;
        coefLabel<nasa9PolyThermo<EquationOfState>::nCoeffs_;
        coefLabel++
    )
    {
        highCpCoeffs[coefLabel] =
            Y1*jt2.highCpCoeffs_[coefLabel]
```

```
            - Y2*jt1.highCpCoeffs_[coefLabel];

        midCpCoeffs[coefLabel] =
            Y1*jt2.midCpCoeffs_[coefLabel]
          - Y2*jt1.midCpCoeffs_[coefLabel];

        lowCpCoeffs[coefLabel] =
            Y1*jt2.lowCpCoeffs_[coefLabel]
          - Y2*jt1.lowCpCoeffs_[coefLabel];
    }

    if
    (
        nasa9PolyThermo<EquationOfState>::debug
     && notEqual(jt2.Tcommon1_, jt1.Tcommon1_)
    )
    {
        FatalErrorInFunction
            << "Tcommon1 " << jt2.Tcommon1_ << " for "
            << (jt2.name().size() ? jt2.name() : "others")
            << " != " << jt1.Tcommon1_ << " for "
            << (jt1.name().size() ? jt1.name() : "others")
            << exit(FatalError);
    }

    if
    (
        nasa9PolyThermo<EquationOfState>::debug
     && notEqual(jt2.Tcommon2_, jt1.Tcommon2_)
    )
    {
        FatalErrorInFunction
            << "Tcommon2 " << jt2.Tcommon2_ << " for "
            << (jt2.name().size() ? jt2.name() : "others")
            << " != " << jt1.Tcommon2_ << " for "
            << (jt1.name().size() ? jt1.name() : "others")
            << exit(FatalError);
    }

    return nasa9PolyThermo<EquationOfState>
    (
        eofs,
        max(jt2.Tlow_, jt1.Tlow_),
        min(jt2.Thigh_, jt1.Thigh_),
        jt2.Tcommon1_,
        jt2.Tcommon2_,
        highCpCoeffs,
        midCpCoeffs,
        lowCpCoeffs
    );
}

// ************************************************************************* //
```

Appendix H Implemented nasaPoly Transport Model

H.1 nasaPolyTransport.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Keivan A.Gh / 15, Nov, 2020:
\*---------------------------------------------------------------------------*/

#include "nasaPolyTransport.H"
#include "IOstreams.H"

// * * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * //

template<class Thermo>
void Foam::nasaPolyTransport<Thermo>::checkInputData() const
{
    if (Tlow_ >= Thigh_)
    {
        FatalErrorInFunction
            << "Tlow(" << Tlow_ << ") >= Thigh(" << Thigh_ << ')'
            << exit(FatalError);
    }

    if (Tcommon_ <= Tlow_)
    {
        FatalErrorInFunction
            << "Tcommon(" << Tcommon_ << ") <= Tlow(" << Tlow_ << ')'
            << exit(FatalError);
    }

    if (Tcommon_ > Thigh_)
    {
        FatalErrorInFunction
            << "Tcommon(" << Tcommon_ << ") > Thigh(" << Thigh_ << ')'
            << exit(FatalError);
    }
}

// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

template<class Thermo>
Foam::nasaPolyTransport<Thermo>::nasaPolyTransport(const dictionary& dict)
:
    Thermo(dict),
```

```
    Tlow_(readScalar(dict.subDict("transport").lookup("Tlow"))),
    Thigh_(readScalar(dict.subDict("transport").lookup("Thigh"))),
    Tcommon_(readScalar(dict.subDict("transport").lookup("Tcommon"))),
    highMuKappaCoeffs_(dict.subDict("transport").lookup("highMuKappaCoeffs")),
    lowMuKappaCoeffs_(dict.subDict("transport").lookup("lowMuKappaCoeffs"))
{
    checkInputData();
}

// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

template<class Thermo>
void Foam::nasaPolyTransport<Thermo>::write(Ostream& os) const
{
     os  << this->name() << endl;
     os  << token::BEGIN_BLOCK  << incrIndent << nl;

     Thermo::write(os);

    dictionary dict("transport");
    dict.add("Tlow", Tlow_);
    dict.add("Thigh", Thigh_);
    dict.add("Tcommon", Tcommon_);
    dict.add("highMuKappaCoeffs",highMuKappaCoeffs_);
    dict.add("lowMuKappaCoeffs",lowMuKappaCoeffs_);

     os  << indent << dict.dictName() << dict;
     os  << decrIndent << token::END_BLOCK << nl;
}

// * * * * * * * * * * * * * * * * Ostream Operator  * * * * * * * * * * * * * //

template<class Thermo>
Foam::Ostream& Foam::operator<<
(
    Ostream& os,
    const nasaPolyTransport<Thermo>& npt
)
{
    npt.write(os);
    return os;
}

// ************************************************************************* //
```

H.2 nasaPolyTransport.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2019 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
```

```
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Keivan A.Gh / 15, Nov, 2020:
Class
    Foam::nasaPolyTransport

Description
    Transport package using NASA polynomial functions for
     \c mu and \c kappa.

Usage

    \table
        Property            | Description
        highMuKappaCoeffs<8>| Dynamic viscosity + Thermal Conductivity polynomial
                              coefficients from Tcommon to Thigh
        lowMuKappaCoeffs<8> | Dynamic viscosity + Thermal Conductivity polynomial
                              coefficients from Tlow to Tcommon
    \endtable

    Example of the specification of the transport properties:
    \verbatim
    transport
    {
        highMuKappaCoeffs<8>     ( 0.70504381, 0.36287686e+02, -0.72255550e+04,
                                   0.41921607, highkappaCoeff, highkappaCoeff,
                                   highkappaCoeff, highkappaCoeff );
        lowMuKappaCoeffs<8>      ( 0.68887644, 0.48727168e+01, -0.59565053e+03,
                                   0.55569577, lowkappaCoeff, lowkappaCoeff,
                                   lowkappaCoeff, lowkappaCoeff );
    }
    \endverbatim

    The polynomial expressions are evaluated as so:

        \f[
           if (T < Tcommon)
          {
                \mu    = exp(0.68887644*log(T) + 0.48727168e+01/T + -
                              0.59565053e+03/T^2 + 0.55569577)*1e-7
          }
          else
          {
                \mu    = exp(0.70504381*log(T) + 0.36287686e+02/T + -
                              0.72255550e+04/T^2 + 0.41921607)*1e-7
          }
        \f]

        \f[
           if (T < Tcommon)
          {
                \kappa    = exp(...)*1e-4
          }
          else
          {
                \kappa    = exp(...)*1e-4
          }
        \f]

Note
    - Dynamic viscosity polynomial coefficients evaluate to an expression in
      [Pa.s].
    - Thermal conductivity polynomial coefficients evaluate to an expression in
      [W/m/K].

SourceFiles
    nasaPolyTransportI.H
```

```
    nasaPolyTransport.C
\*---------------------------------------------------------------------------*/

#ifndef nasaPolyTransport_H
#define nasaPolyTransport_H

#include "scalar.H"
#include "FixedList.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

// Forward declaration of friend functions and operators

template<class Thermo> class nasaPolyTransport;

template<class Thermo>
inline nasaPolyTransport<Thermo> operator+
(
    const nasaPolyTransport<Thermo>&,
    const nasaPolyTransport<Thermo>&
);

template<class Thermo>
inline nasaPolyTransport<Thermo> operator*
(
    const scalar,
    const nasaPolyTransport<Thermo>&
);

template<class Thermo>
Ostream& operator<<
(
    Ostream&,
    const nasaPolyTransport<Thermo>&
);

/*---------------------------------------------------------------------------*\
                          Class nasaPolyTransport Declaration
\*---------------------------------------------------------------------------*/

template<class Thermo>
class nasaPolyTransport
:
    public Thermo
{

public:

    // Public data

        static const int nCoeffs_ = 8;
        typedef FixedList<scalar, nCoeffs_> coeffArray;

private:

    // Private Data

        // Temperature limits of applicability of functions [K]
        scalar Tlow_, Thigh_, Tcommon_;

       // Coefficient arrays consist of both dynamic viscosity and
       // thermal consuctivity coefficients for different temperature range
         coeffArray highMuKappaCoeffs_;
         coeffArray lowMuKappaCoeffs_;

    // Private Member Functions

        //- Check that input data is valid
        void checkInputData() const;
```

```
        //- Return the coefficients corresponding to the given temperature
        inline const coeffArray& coeffs(const scalar T) const;

public:
    // Constructors
        //- Construct from components
        inline nasaPolyTransport
        (
            const Thermo& t,
            const scalar Tlow,
            const scalar Thigh,
            const scalar Tcommon,
            const coeffArray& highMuKappaCoeffs,
            const coeffArray& lowMuKappaCoeffs
        );
        //- Construct as a named copy
        inline nasaPolyTransport(const word&, const nasaPolyTransport&);
       //- Construct from dictionary
        nasaPolyTransport(const dictionary& dict);
       //- Construct and return a clone
        inline autoPtr<nasaPolyTransport> clone() const;
        // Selector from dictionary
        inline static autoPtr<nasaPolyTransport> New(const dictionary& dict);
    // Member Functions
        //- Return the instantiated type name
        static word typeName()
        {
            return "nasaPoly<" + Thermo::typeName() + '>';
        }
        //- Limit the temperature to be in the range Tlow_ to Thigh_
        inline scalar limit(const scalar T) const;
        // Access
            //- Return const access to the low temperature limit
            inline scalar Tlow() const;
            //- Return const access to the high temperature limit
            inline scalar Thigh() const;
            //- Return const access to the common temperature
            inline scalar Tcommon() const;
            //- Return const access to the high temperature poly coefficients
            inline const coeffArray& highMuKappaCoeffs() const;
             //- Return const access to the low temperature poly coefficients
            inline const coeffArray& lowMuKappaCoeffs() const;
        // Fundamental properties
            //- Dynamic viscosity [kg/m/s]
            inline scalar mu(const scalar p, const scalar T) const;
            //- Thermal conductivity [W/m/K]
            inline scalar kappa(const scalar p, const scalar T) const;
            //- Thermal diffusivity of enthalpy [kg/m/s]
            inline scalar alphah(const scalar p, const scalar T) const;
             // Species diffusivity
             // inline scalar D(const scalar p, const scalar T) const;
        // I-O
```

```
            //- Write to Ostream
            void write(Ostream& os) const;

    // Member Operators

        inline void operator+=(const nasaPolyTransport&);
        inline void operator*=(const scalar);

    // Friend operators

        friend nasaPolyTransport operator+ <Thermo>
        (
            const nasaPolyTransport&,
            const nasaPolyTransport&
        );

        friend nasaPolyTransport operator* <Thermo>
        (
            const scalar,
            const nasaPolyTransport&
        );

    // Ostream Operator

        friend Ostream& operator<< <Thermo>
        (
            Ostream&,
            const nasaPolyTransport&
        );
};

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include "nasaPolyTransportI.H"

#ifdef NoRepository
    #include "nasaPolyTransport.C"
#endif

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************************************************************* //
```

H.3 nasaPolyTransportI.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
```

```
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Keivan A.Gh / 15, Nov, 2020:

\*---------------------------------------------------------------------------*/

#include "nasaPolyTransport.H"
#include "specie.H"

// * * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * * //

template<class Thermo>
inline Foam::nasaPolyTransport<Thermo>::nasaPolyTransport
(
    const Thermo& t,
    const scalar Tlow,
    const scalar Thigh,
    const scalar Tcommon,
    const typename nasaPolyTransport<Thermo>::coeffArray& highMuKappaCoeffs,
    const typename nasaPolyTransport<Thermo>::coeffArray& lowMuKappaCoeffs
)
:
    Thermo(t),
    Tlow_(Tlow),
    Thigh_(Thigh),
    Tcommon_(Tcommon)
{
     for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
        {
            highMuKappaCoeffs_[coefLabel] = highMuKappaCoeffs[coefLabel];
            lowMuKappaCoeffs_[coefLabel] = lowMuKappaCoeffs[coefLabel];
        }
}

template<class Thermo>
inline const typename Foam::nasaPolyTransport<Thermo>::coeffArray&
Foam::nasaPolyTransport<Thermo>::coeffs
(
    const scalar T
) const
{
    if (T < Tcommon_)
    {
        return lowMuKappaCoeffs_;
    }
    else
    {
        return highMuKappaCoeffs_;
    }
}

// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //

template<class Thermo>
inline Foam::nasaPolyTransport<Thermo>::nasaPolyTransport
(
    const word& name,
    const nasaPolyTransport& npt
)
:
    Thermo(name, npt),
    Tlow_(npt.Tlow_),
    Thigh_(npt.Thigh_),
    Tcommon_(npt.Tcommon_)
{
    for (label coefLabel=0; coefLabel<nCoeffs_; coefLabel++)
    {
        highMuKappaCoeffs_[coefLabel] = npt.highMuKappaCoeffs_[coefLabel];
```

```
        lowMuKappaCoeffs_[coefLabel] = npt.lowMuKappaCoeffs_[coefLabel];
    }
}

template<class Thermo>
inline Foam::autoPtr<Foam::nasaPolyTransport<Thermo>>
Foam::nasaPolyTransport<Thermo>::clone() const
{
    return autoPtr<nasaPolyTransport<Thermo>>
    (
        new nasaPolyTransport<Thermo>(*this)
    );
}

template<class Thermo>
inline Foam::autoPtr<Foam::nasaPolyTransport<Thermo>>
Foam::nasaPolyTransport<Thermo>::New
(
    const dictionary& dict
)
{
    return autoPtr<nasaPolyTransport<Thermo>>
    (
        new nasaPolyTransport<Thermo>(dict)
    );
}

// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::limit
(
    const scalar T
) const
{
    if (T < Tlow_ || T > Thigh_)
    {
        WarningInFunction
            << "attempt to use nasaPolyTransport<Thermo>"
               " out of temperature range "
            << Tlow_ << " -> " << Thigh_ << ";  T = " << T
            << endl;

        return min(max(T, Tlow_), Thigh_);
    }
    else
    {
        return T;
    }
}

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::Tlow() const
{
    return Tlow_;
}

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::Thigh() const
{
    return Thigh_;
}

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::Tcommon() const
{
    return Tcommon_;
}
```

```
template<class Thermo>
inline const typename Foam::nasaPolyTransport<Thermo>::coeffArray&
Foam::nasaPolyTransport<Thermo>::highMuKappaCoeffs() const
{
    return highMuKappaCoeffs_;
}

template<class Thermo>
inline const typename Foam::nasaPolyTransport<Thermo>::coeffArray&
Foam::nasaPolyTransport<Thermo>::lowMuKappaCoeffs() const
{
    return lowMuKappaCoeffs_;
}

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::mu
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return (exp(a[0]*log(T) + (a[1] + a[2]/T)/T + a[3]))*1e-07;
}

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::kappa
(
    const scalar p,
    const scalar T
) const
{
    const coeffArray& a = coeffs(T);
    return (exp(a[4]*log(T) + (a[5] + a[6]/T)/T + a[7]))*1e-04;
}

template<class Thermo>
inline Foam::scalar Foam::nasaPolyTransport<Thermo>::alphah
(
    const scalar p,
    const scalar T
) const
{
    return kappa(p, T)/this->Cp(p, T);
}

// * * * * * * * * * * * * * * * * Member Operators  * * * * * * * * * * * * * * * //

template<class Thermo>
inline void Foam::nasaPolyTransport<Thermo>::operator+=
(
    const nasaPolyTransport<Thermo>& npt
)
{
    scalar Y1 = this->Y();

    Thermo::operator+=(npt);

    if (mag(this->Y()) > small)
    {
        Y1 /= this->Y();
        scalar Y2 = npt.Y()/this->Y();

        Tlow_ = max(Tlow_, npt.Tlow_);
        Thigh_ = min(Thigh_, npt.Thigh_);

        if
        (
            nasaPolyTransport<Thermo>::debug
```

```
            && notEqual(Tcommon_, npt.Tcommon_)
        )
        {
            FatalErrorInFunction
                << "Tcommon " << Tcommon_ << " for "
                << (this->name().size() ? this->name() : "others")
                << " != " << npt.Tcommon_ << " for "
                << (npt.name().size() ? npt.name() : "others")
                << exit(FatalError);
        }

        for
        (
            label coefLabel=0;
            coefLabel<nasaPolyTransport<Thermo>::nCoeffs_;
            coefLabel++
        )
        {
            highMuKappaCoeffs_[coefLabel] =
                Y1*highMuKappaCoeffs_[coefLabel]
              + Y2*npt.highMuKappaCoeffs_[coefLabel];

            lowMuKappaCoeffs_[coefLabel] =
                Y1*lowMuKappaCoeffs_[coefLabel]
              + Y2*npt.lowMuKappaCoeffs_[coefLabel];
        }
    }
}

template<class Thermo>
inline void Foam::nasaPolyTransport<Thermo>::operator*=
(
    const scalar s
)
{
    Thermo::operator*=(s);
}

// * * * * * * * * * * * * * * * * Friend Operators  * * * * * * * * * * * * * //

template<class Thermo>
inline Foam::nasaPolyTransport<Thermo> Foam::operator+
(
    const nasaPolyTransport<Thermo>& npt1,
    const nasaPolyTransport<Thermo>& npt2
)
{
    Thermo t
    (
        static_cast<const Thermo&>(npt1)
      + static_cast<const Thermo&>(npt2)
    );

    if (mag(t.Y()) < small)
    {
        return nasaPolyTransport<Thermo>
        (
            t,
             0,
            npt1.Tlow_,
            npt1.Thigh_,
            npt1.Tcommon_,
            npt1.highMuKappaCoeffs_,
            npt1.lowMuKappaCoeffs_
        );
    }
    else
    {
```

```
        scalar Y1 = npt1.Y()/t.Y();
        scalar Y2 = npt2.Y()/t.Y();

        typename nasaPolyTransport<Thermo>::coeffArray highMuKappaCoeffs;
        typename nasaPolyTransport<Thermo>::coeffArray lowMuKappaCoeffs;

        for
        (
            label coefLabel=0;
            coefLabel<nasaPolyTransport<Thermo>::nCoeffs_;
            coefLabel++
        )
        {
            highMuKappaCoeffs[coefLabel] =
                Y1*npt1.highMuKappaCoeffs_[coefLabel]
              + Y2*npt2.highMuKappaCoeffs_[coefLabel];

            lowMuKappaCoeffs[coefLabel] =
                Y1*npt1.lowMuKappaCoeffs_[coefLabel]
              + Y2*npt2.lowMuKappaCoeffs_[coefLabel];
        }

        if
        (
            nasaPolyTransport<Thermo>::debug
         && notEqual(npt1.Tcommon_, npt2.Tcommon_)
        )
        {
            FatalErrorInFunction
                << "Tcommon " << npt1.Tcommon_ << " for "
                << (npt1.name().size() ? npt1.name() : "others")
                << " != " << npt2.Tcommon_ << " for "
                << (npt2.name().size() ? npt2.name() : "others")
                << exit(FatalError);
        }

        return nasaPolyTransport<Thermo>
        (
            t,
            max(npt1.Tlow_, npt2.Tlow_),
            min(npt1.Thigh_, npt2.Thigh_),
            npt1.Tcommon_,
            highMuKappaCoeffs,
            lowMuKappaCoeffs
        );
    }
}

template<class Thermo>
inline Foam::nasaPolyTransport<Thermo> Foam::operator*
(
    const scalar s,
    const nasaPolyTransport<Thermo>& npt
)
{
    return nasaPolyTransport<Thermo>
    (
        s*static_cast<const Thermo&>(npt),
        npt.Tlow_,
        npt.Thigh_,
        npt.Tcommon_,
        npt.highMuKappaCoeffs_,
        npt.lowMuKappaCoeffs_
    );
}

// ************************************************************************* //
```

Appendix I The thermo.compressibleGas Dictionary

This dictionary contains the last version of the generated coefficients for the implemented *nasa9Poly* thermodynamic model and *nasaPoly* transport model.

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*\
     Keivan A.Gh                          | November 2020
                                          | Version: 1
     These Coefficients are based on NASA 7 transport polynomials and NASA 9
     thermo polynomials, in combination with specific ideal gas equation of
     state for each of these species. (H2: Leachman, O2:Woolly, N2:Span)
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermo.compressibleGas;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
O2
{
    specie
    {
        molWeight          31.99880;
    }
    thermodynamics
    {
        Tlow               10;
        Thigh              6000;
        Tcommon1           300;
        Tcommon2           1000;
        highCpCoeffs       (-1.037939022e+06 2.344830282e+03 1.819732036
                             1.267847582e-03 -2.188067988e-07 2.053719572e-11
                            -8.193467050e-16 -1.689010929e+04 1.738716506e+01 );
        midCpCoeffs        (-3.425563420e+04 4.847000970e+02 1.119010961
                             4.293889240e-03 -6.836300520e-07 -2.023372700e-09
                             1.039040018e-12 -3.391454870e+03 1.849699470e+01 );
        lowCpCoeffs        (-0.17648260350863987 0.6683783566972065
                             3.4751127770152572 0.0004090685462294075
                            -2.7857436034490034e-06 6.673120289909645e-09
                             6.120975676491904e-13 -1046.9564617109088
                             4.811436023617842 );
    }
    transport
    {
        Tlow               90.2;
        Thigh              6000;
        Tcommon            1000;
      highMuKappaCoeffs    ( 0.63839563 -0.12344438e+01 -0.22885810e+05
                             0.18056937e+01 0.80805788 0.11982181e+03
                            -0.47335931e+05 0.95189193 );
      lowMuKappaCoeffs     ( 0.6148139161181673 -55.019686999527096
                             710.3199361179903 2.0000703788706247 0.772734006169922
                            -49.134046555034466 395.2853478841008 1.3145444577293 83
                            );
```

```
    }
    elements
    {
        O                 2;
    }
    equationOfState
    {
        Tc                154.6;
        Vc                73.4e-03;
        Zc                0.288;
        Pc                50.43e05;
        omega             0.022;
    }
}

H2
{
    specie
    {
        molWeight         2.01588;
    }
    thermodynamics
    {
        Tlow              50;
        Thigh             6000;
        Tcommon1          300;
        Tcommon2          1000;
        highCpCoeffs      ( 5.608128010e+05 -8.371504740e+02 2.975364532
                            1.252249124e-03 -3.740716190e-07 5.936625200e-11
                           -3.606994100e-15 5.339824410e+03 -2.202774769 );
        midCpCoeffs       ( 4.078323210e+04 -8.009186040e+02 8.214702010
                           -1.269714457e-02 1.753605076e-05 -1.202860270e-08
                            3.368093490e-12 2.682484665e+03 -3.043788844e+01 );
       lowCpCoeffs        ( 181.9135344883896 27.079531981481814 1.3104682284562241
                            0.011220776470717864 1.611217017433592e-05
                           -1.9240810285965655e-07 3.0193152336728514e-10
                           -947.6966125079408 5.383975661362266 );
    }
    transport
    {
        Tlow               20.4;
        Thigh              6000;
        Tcommon            1000;
        highMuKappaCoeffs  ( 0.70504381 0.36287686e+02 -0.72255550e+04 0.41921607
                             0.74368397 -0.54941898e+03 0.25676376e+06
                             0.35553997e+01 );
        lowMuKappaCoeffs   ( 0.6756835522136462 -2.871663879485968
                           -51.92244186666171 0.6527632646934771 0.6999799625099263
                           -26.239569737479016 308.1554996341592 3.588781793251602
                           );
    }
    elements
    {
        H                 2;
    }
    equationOfState
    {
        Tc                33.19;
        Vc                64.1e-03;
        Zc                0.305;
        Pc                13.13e05;
        omega             -0.216;
    }
}

N2
```

```
{
    specie
    {
        molWeight          28.01340;
    }
    thermodynamics
    {
        Tlow               20;
        Thigh              6000;
        Tcommon1           300;
        Tcommon2           1000;
        highCpCoeffs       ( 5.877124060e+05 -2.239249073e+03 6.066949220
                             -6.139685500e-04 1.491806679e-07 -1.923105485e-11
                              1.061954386e-15 1.283210415e+04 -1.586640027e+01 );
        midCpCoeffs        ( 2.210371497e+04 -3.818461820e+02 6.082738360
                             -8.530914410e-03 1.384646189e-05 -9.625793620e-09
                              2.519705809e-12 7.108460860e+02 -1.076003744e+01 );
        lowCpCoeffs        ( 1.3915354401621036 -0.16209310172173885
                              3.5066025887212535 -0.00011438881321350624
                              9.974493397669226e-07 -3.8988785307732025e-09
                              5.764382510995432e-12 -1043.3066964252632
                              3.0742897156394626 );
    }
    transport
    {
        Tlow               77.3;
        Thigh              6000;
        Tcommon            1000;
        highMuKappaCoeffs  ( 0.65060585 0.28517449e+02 -0.16690236e+05
                              0.15223271e+01 0.65147781 -0.15059801e+03
                             -0.13746760e+05 0.21801632e+01 );
        lowMuKappaCoeffs   ( 0.570825726949261 -73.15782924390622 1731.509463097786
                              2.1561587209523765 0.795646255032684
                             -1.2864274422960777 -881.7940891215968
                              1.0196220007343342 );
    }
    elements
    {
        N                  2;
    }
    equationOfState
    {
        Tc                 126.2;
        Vc                 89.2e-03;
        Zc                 0.289;
        Pc                 34.00e05;
        omega              0.038;
    }
}

// ************************************************************************* //
```