Cite as: Meier, R.: Implementation of FGM model for premixed flames in OpenFOAM. In Proceedings of CFD with OpenSource Software, 2022, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2022

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of FGM model for premixed flames in OpenFOAM

Developed for OpenFOAM-v2112

Author:
Rafael Meier
Federal University of Santa Catarina, Brazil

Peer reviewed by:
Örjan Fjällborg
Mohammad Khanouki
Thiago Souza

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 23, 2023

Learning outcomes

The main requirements of a tutorial in the course is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- how to solve a canonical combustion problem using a Flamelet-Generated Manifold (FGM) method for premixed flames.

The theory of it:

- the theory of combustion modeling and the theory of FGM for premixed flames.

How it is implemented:

- how the thermochemical properties (density, temperature, reaction rate, etc.) from an OpenFOAM default solver are replaced by tabulated properties previously computed following the FGM methodology.
- how to implement a transport equation for the combustion variables.

How to modify it:

- how the implementation is done from the rhoReactingBuoyantFoam solver. A database considering a stochiometric combustion of methane/air will be provided, showing how the user can set the appropriate manifold database. In addition, a simulation case in the Bunsen burner will be presented, where the user can modify the inlet velocity profile as needed.

Prerequisites

The reader is expected to know in advance the following topics in order to get maximum benefit out of this report:

- The fundamentals of fluid mechanics and combustion.
- Familiarity with CFD.
- Basic knowledge of object orientation and C++ syntax.
- Experience to run standard tutorials in OpenFOAM and usage of Paraview.

Contents

Nomenclature

Roman symbols

c	progress variable
c_p	isobaric heat capacity
$\mathcal{D}$	mass diffusivity
$D_{m,i}$	diffusion coefficient of the i component into the mixture
h	total sensible enthalpy
h_i^o	enthalpy of formation of species i
$j_i^{''}$	diffusive mass flux
Ka	Karlovitz number
l	characteristic length
Le	Lewis number
MW	molecular weight
p	pressure
Pr	Prandtl number
$\dot{Q}^{''}$	heat flux
$\dot{Q}^{'''}$	volumetric heat release rate
R_u	universal gas constant
S_{cf}	cell face area
S_d	flame displacement speed
$\tilde{S}_d$	density-weighted flame displacement speed
S_l^0	flat flame speed
S_l	flame speed
Sc	Schmidt number
t	time
T	temperature
$\vec{V}$	velocity
$\vec{V}_f$	flame velocity
V	volume
$u^{'}$	turbulence velocity
Y_i	mass fraction of component i
$\vec{z}$	high vector

Greek symbols

α	thermal diffusivity
δ_l	flame thickness
δ_r	reacting zone thickness
η	Kolmogorov length scale
λ	heat conductivity
μ	viscosity
ν	momentum diffusivity
ρ	density

τ	characteristic time
ψ	compressibility factor
$\dot{\omega}_i'''$	reaction rate

Abbreviations

CFD	Computional Fluid Dynamics
CSP	Computational Singular Perturbation model
DNS	Direct Numerical Simulation
FGM	Flamelet-Generated Manifold model
ILDM	Intrinsic Low-Dimensional Manifolds model
OpenFOAM	Open source Field Operation And Manipulation

Subscripts

b	burned gas
c	related to the progress variable
$chem$	chemical
f	flame
cf	cell face
g	gas
i	species i
m	mixture-averaged
nb	neighbor cells
P	cell center point
t	turbulent
u	unburned gas

Chapter 1

Theory

1.1 Introduction

Combustion physics is a complex phenomenon based upon the competition of flow and chemistry time scales of varying magnitudes. In terms of computational fluid dynamics (CFD) modeling, roughly speaking, the models decouple the set of partial differential equations governing the flow field from the system of ordinary differential equations of chemical kinetics. This is justified since, in general, the time scales of the chemical reactions are much faster than the characteristic time scale of the flow. However, reactions of different orders of magnitude exist even in case of simplified chemical mechanisms, resulting in the mathematical stiffness problem.

When a fundamental research approach of the phenomena is considered, e.g., computational modeling based on Direct Numerical Simulations (DNS), it is necessary to use all physical descriptions to understand the intrinsic details of the interaction involving the flow-chemistry process. However, in terms of engineering applications, i.e., gas turbines for electrical power generation, aircraft engines, and internal combustion engines for cars, trucks, ships, and furnaces, the domains for simulations are substantially enlarged, becoming impractical the utilization of detailed models for transport equations and chemical kinetics. In literature, numerical models have focused on decreasing the computational complexity in order to make feasible combustion studies using regular computational architectures. Chemical reduction techniques are one of the ways to tackle this problem. Some of them are referred as conventional reduction [1], Intrinsic Low-Dimensional Manifolds (ILDM) [2], Computational Singular Perturbation (CSP) [3], and the Flamelet-Generated Manifold model (FGM) [4]. These models are based on the idea that most chemical time scales are very small compared to the flow time scales. If all transport processes are neglected, a time-scale analysis can be performed, and the fastest time scales are assumed to be at steady-state. Mathematically, this means that all variables can be stored in a database as a function of a few controlling variables, and during the run-time, only the equations for the controlling variables are solved [5]. In this work, an implementation of the FGM model in OpenFOAM's CFD platform will be presented.

1.2 Laminar premixed flames structure and propagation speed

Figure 1.1 shows a premixed flame front propagating embedded in a flow field $\vec{V}$ [6]. A magnified structure across the flame front may be viewed where it is presented the flame surface separating the burned and unburned mixture, a flame sheet approximation, and the laminar flame structure, which are the basis for a thin flame approximation [7, 8, 9]. Represented by $Y_{F,u}$, the mass fractions of the reactants in the unburned gas are constant until the flame front, where they begin to be consumed, and products begin to form as the temperature increases monotonically from the unburnt region, T_u, to the burnt region T_b. The flame structure reveals the flame preheating zone, with a characteristic length scale δ_l, and a thin reaction zone, with a characteristic length scale $\delta_r \ll \delta_l$.

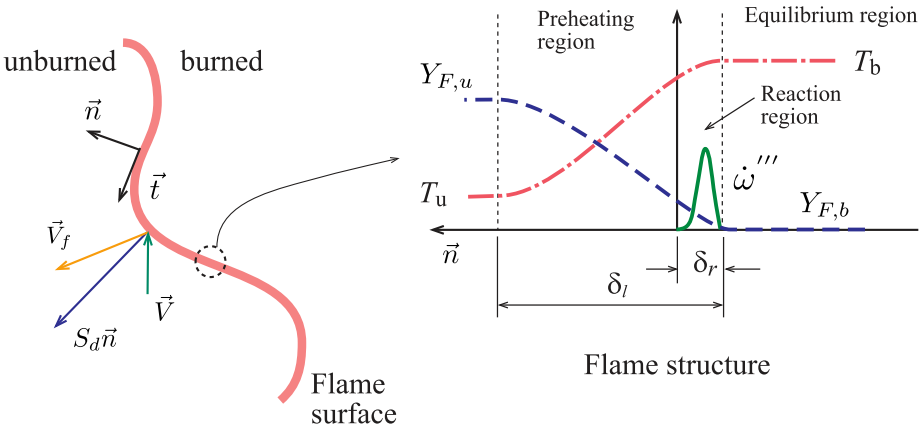


Figure 1.1: Flame sheet approximation and laminar flame structure [6].

Naming S_d the displacement speed and $\vec{V}$ the flow velocity of the unburned mixture in the vicinity of the flame surface, the local propagation velocity of the flame surface $\vec{V}_f$ is

$$\vec{V}_f = \vec{V} + S_d\vec{n} \tag{1.1}$$

where $S_d\vec{n}$ is the displacement velocity and $\vec{n}$ is the normal unit vector at the flame surface, pointed to the unburned mixture side, Fig. 1.1. The displacement speed measures the front speed relative to the difference between the flow speed and the front speed $\vec{V}_f$ [8]

$$S_d = (\vec{V}_f - \vec{V}) \cdot \vec{n}. \tag{1.2}$$

For a meaningful comparison between values of the displacement speed defined at different locations, S_d is often normalized by the ratio of local density ρ to the density if the fresh mixture ρ_u yielding

$$\tilde{S}_d = \frac{\rho}{\rho_u} S_d. \tag{1.3}$$

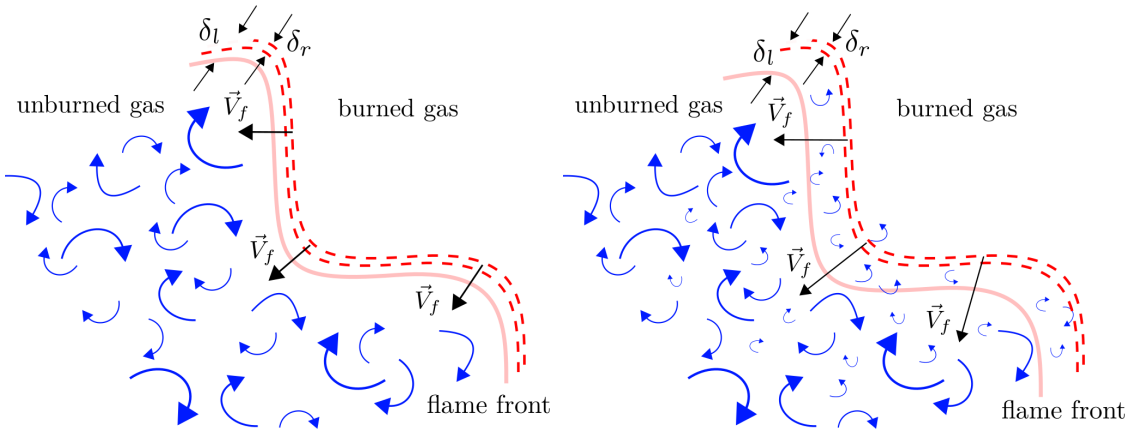
The density-weighted flame displacement speed, $\tilde{S}_d$, can be directly compared to the laminar flame speed S_l for the same reactant mixture [10]. When a flame propagates freely in one-dimensional adiabatic domain in a steady-state regime, it is called the laminar free flame. Its propagating speed is named the flat flame speed $S_l^0 = S_l$ where $\vec{V}_f = 0$, and it is used as a reference quantity for premixed combustion analysis.

1.3 Flamelet-Generated Manifold model

Laminar flamelet methods are based on the assumption that flame structures are much thinner than most length scales of the flow, also implying that the chemical reactions are very fast compared to all other time scales. For that reason, the internal structure of the flame front is almost frozen while it moves around the embedded flow field. The dynamics of the thin flame front is then determined by using a kinematic equation for the propagation of the flame front, in case of non-premixed or partially premixed flames, the mixture fraction equation for the mixing should also be considered, and a CFD solver to solve the conservation equations related to the flow [5, 11].

Figures 1.2 and 1.3 illustrate the premises of the flamelet modeling. In many combustion applications, the flamelet concept plays a key role. The flamelet is a one-dimensional element of the flame front with the front structure of a laminar flame. The flamelet concept is connected with the fact that in most premixed situations, the structure of the flame will depend only on the normal direction of the flame front. In general, the reactive front is very thin compared to the other flame scales [8].

This so-called flamelet hypothesis states that the flame front will retain its one-dimensional laminar structure as it propagates in a turbulent flow [12]. Figure 1.2 points out this feature, showing that for wrinkled flame regimes and even for the corrugated regime, the vector flame speed, $\vec{V}_f$, remains along the direction normal to the flame front area despite the distortions and effects that a turbulent flow may cause.



(a) wrinkled flamelet regime (b) corrugated regime

Figure 1.2: Premixed flame propagating in a turbulent flow.

The Borghi-Peters diagram for premixed turbulent combustion, presented in Fig. 1.3, places the theoretical assumptions in terms of flow and chemistry time scales. The diagram shows how turbulence affects the flame propagating speed and the internal structure of the flame through the normalized relations of turbulence velocity $u^{'}$ and flame speed S_l^0, and the characteristic length scales of turbulent l_t and flame thickness the δ_l.

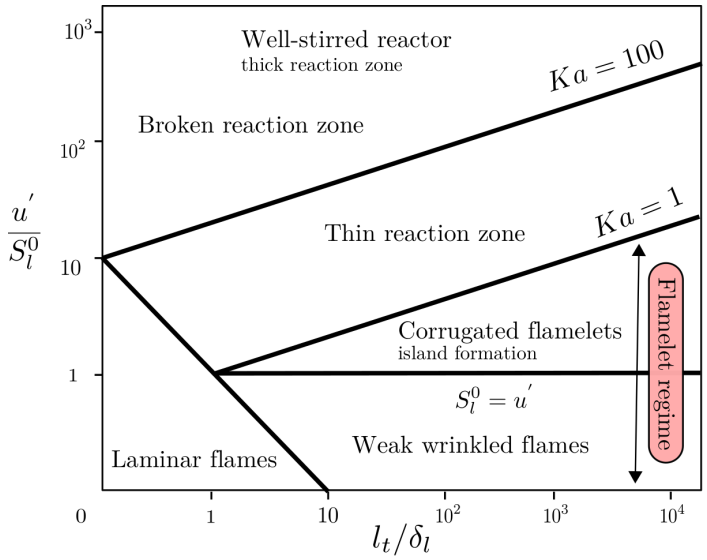


Figure 1.3: Borghi-Peters regime diagram for premixed turbulent combustion.

A general relation is called turbulent Karlovitz number, and it is defined as

$$\mathrm{Ka} = \frac{\tau_{chem}}{\tau_\eta} \tag{1.4}$$

where τ_{chem} and τ_η are the chemistry and Kolmogorov time scales, respectively. Equation (1.4) can be expanded in terms of flame thickness δ_l comparing the smallest turbulent scales, i.e., the so-called Kolmogorov scales η, Eq. (1.5),

$$\mathrm{Ka} = \left(\frac{\delta_l}{\eta}\right)^2 = \left(\frac{l_t}{\delta_l}\right)^{-\frac{1}{2}} \left(\frac{u^{'}}{S_l}\right)^{\frac{3}{2}}. \tag{1.5}$$

Therefore, the accuracy of the flamelet approximation relies on conditions where the turbulent intensity does not substantially affect the innner flame structure which remains close to a laminar flame, wrinkled by turbulence motions, $\mathrm{Ka} \leq 1$ in Eq. (1.5). In this case, the mean burning rate may be estimated from the burning rate of a laminar flame multiplied by the overall flame surface [8].

In the FGM framework, a database representing the combustion process is initially built by storing a set of laminar one-dimensional flames directly solved with detailed chemical kinetic (flamelets) as a function of one (or a few) reaction control variables. FGM reduces the number of equations to be solved and reduces the stiffness of the system of equations [13, 14]. In some cases, this methodology can be a hundred times faster, for instance, than the direct integration of the conservation equations [15] without losing much accuracy. It is important to notice that according to Fig. 1.3, the present model developed here is limited to the weak wrinkled flamelet regime or, in other words, to conditions where the turbulence is $u^{'} \leq S_l^0$.

1.4 Governing equations and computational arrangement

For a detailed description of a reacting flow, the system of governing equations (assuming ideal gases and perfect mixtures) is as follows [8, 16, 17]. The conservation of total mass is defined as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{V} = 0 \tag{1.6}$$

where ρ is the density, t is time and $\vec{V}$ the fluid velocity vector. The conservation of momentum is given by

$$\frac{\partial \rho \vec{V}}{\partial t} + \nabla \cdot \rho \vec{V}\vec{V} = -\nabla p + \nabla \cdot \bar{\bar{\tau}} + \rho \vec{g} \tag{1.7}$$

in which p is the pressure, $\vec{g}$ is the gravitational acceleration. The viscous stress tensor $\bar{\bar{\tau}}$ is computed for a Newtonian fluid using the Stokes assumption

$$\bar{\bar{\tau}} = \mu \left(\nabla \vec{V} + (\nabla \vec{V})^T - \frac{2}{3} \bar{\bar{I}} \nabla \cdot \vec{V} \right) \tag{1.8}$$

where $\bar{\bar{I}}$ is the identity tensor and μ the dynamic viscosity. In a reacting flow, the conservation of mass for species may be expressed as

$$\frac{\partial \rho Y_i}{\partial t} + \nabla \cdot \rho(\vec{V} + \vec{V}_c) Y_i = \dot{\omega}_i^{'''} - \nabla \cdot j_i^{''}, \quad i = 1, .., N-1. \tag{1.9}$$

Y_i is the mass fraction species i and $\dot{\omega}_i^{'''}$ its reaction rate. N is the number of species. The correction velocity $\vec{V}_c$ force the sum of all diffusive fluxes $j_i^{''}$ to be zero

$$\vec{V}_c = -\frac{1}{\rho} \sum_{i=1}^{N} j_i^{''}. \tag{1.10}$$

The diffusive mass flux is computed using a mixture-averaged model derived from the Stefan-Maxwell equations [18], neglecting pressure and temperature diffusion (Soret effect)

$$j_i^{''} = -\rho D_{m,i} \nabla Y_i. \tag{1.11}$$

$D_{m,i}$ is the diffusion coefficient of species i in the mixture.

The conservation of energy is built from the total enthalpy

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot \rho \vec{V} h = -\nabla \cdot \dot{Q}'' + \frac{\partial p}{\partial t} - \sum_{i=1}^{N} h_i^o \dot{\omega}_i''' . \tag{1.12}$$

The transport of energy is formulated in terms of the total sensible enthalpy $h = h_s + \frac{1}{2}\vec{V} \cdot \vec{V}$ and

$$-\nabla \cdot \dot{Q}'' = \underbrace{\nabla \cdot \left(\frac{\lambda}{c_p} \nabla h_s \right) - \sum_{i=1}^{N} \nabla \cdot \left(\frac{\lambda}{c_p} h_{s,i} \nabla Y_i \right)}_{\nabla \cdot (\lambda \nabla T)} - \sum_{i=1}^{N} \nabla \cdot \left(h_{s,i} \hat{j}_i'' \right) \tag{1.13}$$

where λ is the heat conductivity of the mixture, c_p the isobaric heat capacity and T the temperature. Viscous work, potential energy, radiation, and Dufour effect are neglected. The sensible enthalpy $h_{s,i}$ of species i and the sensible enthapy of the mixture h_s for ideal gases is given by

$$h_{s,i} = h_i - h_i^o, \quad h_s = \sum_{i=1}^{N} Y_i h_{s,i}, \tag{1.14}$$

where $h_i^o = h_i(298\ \mathrm{K})$ is the enthalpy of formation of species i. The corrected diffusive mass flux $\hat{j}_i''$ is given by

$$\hat{j}_i'' = j_i'' - Y_i \sum_{i=1}^{N} j_i'' . \tag{1.15}$$

The Fouries's second law $\nabla \cdot (\lambda \nabla T)$ is rewritten (assuming ideal gases and that all species have the same temperature) in order to obtain the first term on the r.h.s. of Eq. (1.13), which can is discretized implicitly. The equation for the ideal gases closes the system of partial differential equations

$$\rho = \frac{p \mathrm{MW}_g}{R_u T}, \tag{1.16}$$

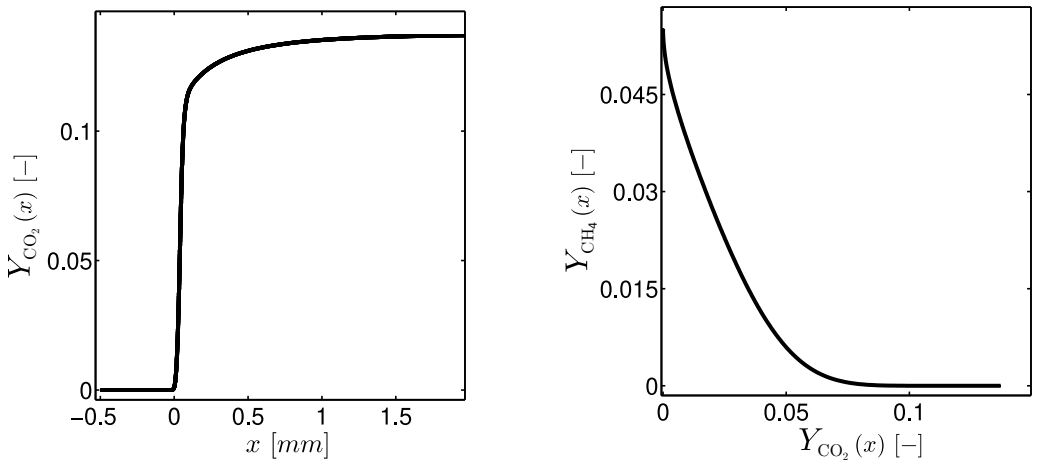
where $\mathrm{MW_g}$ is the molar mass of the mixture and R_u the universal gas constant.

1.4.1 FGM procedure and numerical structure

Based on the flamelet assumption, in the FGM model the set of combustion thermochemistry variable, e.g., temperature, density and species concentrations are parameterized as a function of specific variables controlling the temporal evolution of the combustion process. A look-up procedure with any CFD code can effectively be done to retrieve the thermochemistry variables from the FGM generated chemistry database [19]. Here, the flamelet database is obtained from a steady one-dimensional, strechless, freely propagating laminar flame solved with the CHEM1D package [20]. The solution is performed for a stoichiometric premixed CH_4/air flame at atmospheric conditions. The chemical mechanism used to generat the manifold was the well-stablished GRI3.0 mechanism with 53 species and 325 elementary reactions [21].

The flamelet manifold is based on the CO_2 species as the reaction progress, c, which controls the evolution of the combustion process since it presents a monotonic evolution as seen in Fig. 1.4a [12, 19]. The set of thermo-chemistry database, i.e., temperature, density and reaction rate, are tabulated as a function of the progress variable chosen. As an example of the tabulation procedure, Fig. 1.4b shows the mass fraction of methane, $Y_{\mathrm{CH_4}}$, as a function of the non-scaled progress variable, Y_{CO2}.

This case can be fully described using only carbon dioxide as the reaction progress variable. Similar conclusions are establish for other thermo-chemistry quantities. By scaling the reaction



(a) Mass fraction of CO_2 as function of x-axis.

(b) Mass fraction of CH_4 as function of the non-scaled progress variable, i.e., Y_{CO_2}.

Figure 1.4: Premixed flame structure computed with CHEM1D and GRI3.0 reaction mechanism [12, 19].

progress variable to vary between $c = 0$ and $c = 1$, the state of the combustion process is then completely identified, such that at $c = 0$ the mixture is in the unburned state and at $c = 1$ in the fully burnt state. The scaled progress variable c is defined as follows

$$c = \frac{Y_{CO_2} - \min(Y_{CO_2})}{\max(Y_{CO_2}) - \min(Y_{CO_2})}, \tag{1.17}$$

where $\min(Y_{CO_2})$ and $\max(Y_{CO_2})$ are taken from the lower and the upper limit of $Y_{CO_2}(x)$, as shown in Fig. 1.4a. To identify easily the state of the combustion process, the computed database is tabulated as a function of the scaled progress variable, Eq. (1.17). The manifold generated is interpolated on a 1D equidistant grid with 801 points lineary distributed between 0 and 1. Figures 1.5 show the premixed laminar database using the FGM approach for the methane/air premixed flame.

The local progress of combustion is given only in terms of a single control variable, i.e., c, by tracking the evolution of the scalar c through a transport equation, such that the entire combustion process is immediately retrived within the FGM approximation. Figure 1.6 shows the interface procedure between the FGM-CFD algorithm for direct simulations. The CFD executes a lookup procedure on FGM pre-processing database updating the thermo-chemistry states as the transport equation for c is solved in the CFD code.

The set of transport equation for the variable c and the thermo-chemistry states are defined as

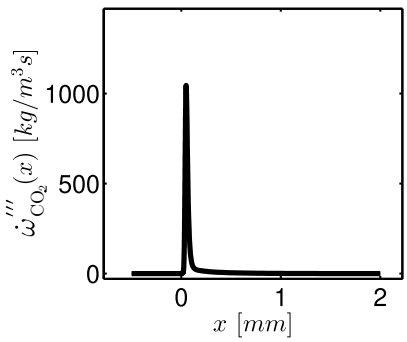
$$\frac{\partial(\rho c)}{\partial t} + \nabla \cdot \rho \vec{V} c = \nabla \cdot \rho \mathcal{D}_c \nabla c + \dot{\omega}_c^{'''} \tag{1.18}$$

$$\rho = f_1(c) \tag{1.19}$$

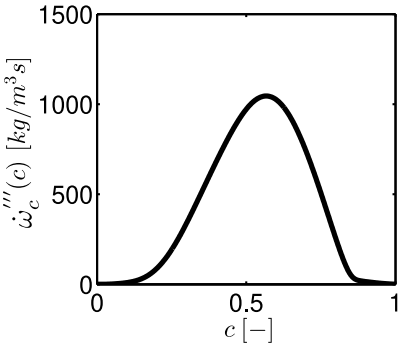
$$T = f_2(c) \tag{1.20}$$

$$\dot{\omega}_c^{'''} = f_3(c) \tag{1.21}$$

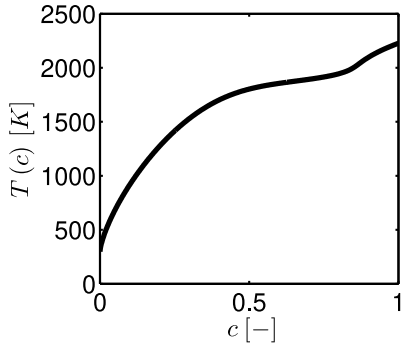
Now, that the entire complexity observed in the Equations determining the species transport, Eq. (1.9) and energy, Eq. (1.12), is replaced by a single transport Eq. (1.18) with the lookup procedure proving the corresponding values of ρ, T and $\dot{\omega}_c^{'''}$.



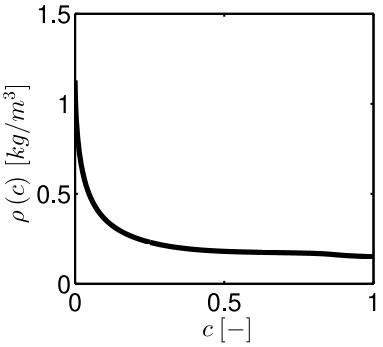
(a) Source term of CO_2 as function of x-axis in physical space.



(b) Source term of CO_2 as function of the scaled progress variable, c.



(c) Temperature, T, as a function of the progress variable.



(d) Density of the mixture, ρ, as a function of the progress variable.

Figure 1.5: Premixed flame structure computed with CHEM1D and GRI3.0 reaction mechanism [12, 19].

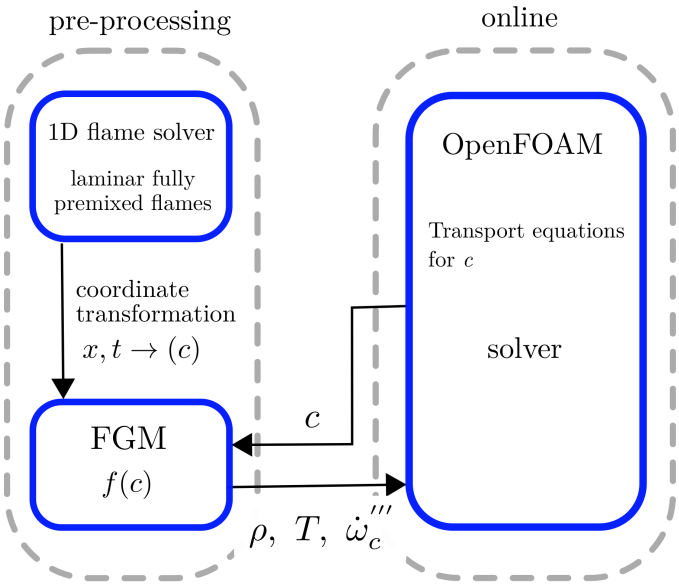


Figure 1.6: Implementation of FGM as the combustion model in a CFD code: DNS case.

The dynamic viscosity μ is a function of temperature and modeled using the Sutherland's law. The diffusion coefficient $\mathcal{D}_c$ in Eq. (1.18) is modeled considering the ratio between the temperature diffusivity $\rho\mathcal{D}$ and the Lewis number of the progress variable chosen, i.e., $\mathrm{Le}_{\mathrm{CO_2}}$, [22, 23]

$$\rho\mathcal{D} = \frac{\lambda}{c_p} \tag{1.22}$$

$$\mathcal{D}_c = \frac{\lambda}{c_p \mathrm{Le}_{\mathrm{CO_2}}} \tag{1.23}$$

where $\mathrm{Le}_{\mathrm{CO_2}} = 1.38$ and the mass diffusivity term is based on [22]

$$\frac{\lambda}{c_p} = 2.58 \cdot 10^{-5} \left(\frac{T}{298} \right)^{0.69}. \tag{1.24}$$

Chapter 2

Reacting flow solver

This chapter provides a top-level description of the `rhoReactingBuoyantFoam`, a native OpenFOAM solver that comes as an optional platform in `reactingFoam`. The application is a solver for chemical reactions using a density-based thermodynamics package with enhanced buoyancy treatment. The majority of compressible solvers implemented in OpenFOAM use the PIMPLE algorithm, which merges the controls of PISO and SIMPLE pressure correction schemes, providing characteristics of a full-transient solver or resulting in a pseudo-transient simulation [24].

The algorithm solves the compressible fluid equation based on the pressure correction equation (similar to incompressible flows), which establishes the physical connection between the momentum and the continuity equation. The chemical species and energy transport equations are part of the algorithm's linear sequence since their results directly influence the mass flux through the mesh cells. The internal time step loop is solved as follows in List. 2.1

1. Solve the density equation, ρ, `rhoEqn.H`.
2. Solve the momentum equation, $\vec{V}$, `UEqn.H`, where it generates a temporary velocity fields $\vec{V}^*$, that don't satisfy the continuity equation before the restriction imposed by the pressure gradient.
3. Solve the transport for the chemical species equation, $Y_1, ..., Y_N$, `YEqn.H`, where N is the number of species.
4. Solve the energy transport equation, defined in terms of enthalpy, `EEqn.H`. The species and energy equation are connected by the energy heat release rate, $\dot{Q}'''$, and the temperature field comes out from the enthalpy solution. In addition, pressure and temperature are key variables used to update the thermodynamic variables, e.g., the equation of state to compute the density and the transport variables.
5. Solve the pressure equation, p, `pEqn.H`. To ensure mass conservation, the continuity and the momentum equation are coupled through the pressure equation (and with the equation of state) which satisfies the continuity equation. Also a correction for the velocity is computed from pressure $p = p^* + p'$ and $\vec{V} = \vec{V}^* + \vec{V}'$, where $\vec{V}' = f(p')$, until it satisfies the mass conservation.
6. Correct the density through the new pressure and temperature field using the equation of state.

Listing 2.1: `reactingFoam/rhoReactingBuoyantFoam/rhoReactingBuoyantFoam.C`

```
101 #include "rhoEqn.H"
102
103         // --- Pressure-velocity PIMPLE corrector loop
104         while (pimple.loop())
105         {
106             #include "UEqn.H"
107             #include "YEqn.H"
108             #include "EEqn.H"
109
110             // --- Pressure corrector loop
111             while (pimple.correct())
112             {
113                 #include "pEqn.H"
114             }
115
116             if (pimple.turbCorr())
117             {
118                 turbulence->correct();
119             }
120         }
121
122         rho = thermo.rho();
123         }
```

2.1 Mass conservation

The `rhoReactingBuoyantFoam` is a solver that uses the variable-density continuity equation as shown in Eq. (1.6). It is solved following List. 2.2

Listing 2.2: `src/finiteVolume/cfdTools/compressible/rhoEqn.H`

```
1  {
2      fvScalarMatrix rhoEqn
3      (
4          fvm::ddt(rho)
5        + fvc::div(phi)
6        ==
7          fvOptions(rho)
8      );
9
10     fvOptions.constrain(rhoEqn);
11
12     rhoEqn.solve();
13
14     fvOptions.correct(rho);
15 }
```

In OpenFOAM, the `fvm` class stands for finite-volume matrix, and it is used when operations are to be implicit and a left-hand side matrix is formed. This is opposed to the `fvc` class, which stands for finite-volume calculus, and used for explicit operations, such as forming the right-hand side of the matrix equation [25]. The `fvm::ddt(rho)` term is the time derivative of density. In line 5, `fvc::div(phi)` represents the divergence of the mass flux $\rho\vec{V}$.

2.2 Momentum conservation

The equations of motion are given by the composition of Eqs. (1.7) and (1.8) yielding

$$\frac{\partial \rho\vec{V}}{\partial t} + \nabla\cdot\rho\vec{V}\vec{V} = -\nabla p + \rho\vec{g} + \nabla\cdot\left(2\mu_{eff}\boldsymbol{D}(\vec{V})\right) - \nabla\left(\frac{2}{3}\mu_{eff}(\nabla\cdot\vec{V})\right), \tag{2.1}$$

where p is the static pressure field. The effective viscosity μ_{eff} is the sum of the molecular and turbulent viscosity and the rate of strain (deformation) tensor, $\boldsymbol{D}(\vec{V})$, is defined as $\boldsymbol{D}(\vec{V}) = \frac{1}{2}\left(\nabla\vec{V} + (\nabla\vec{V})^T\right)$ as seen in Eq. (1.8).

In the OpenFOAM implementation, the solvers that consider buoyant effects define the pressure solution in terms of p_{rgh} which stands for the pressure without the hydrostatic pressure

$$p_{rgh} = p - \rho\vec{g}\cdot\vec{z} \tag{2.2}$$

where $\vec{z}$ is a positive vector. In this way, the pressure gradient and gravity force terms are rearranged according to

$$\nabla p + \rho\vec{g} = -\nabla(p_{rgh} + \rho\vec{g}\cdot\vec{z}) + \rho\vec{g}, \tag{2.3}$$

$$\nabla p + \rho\vec{g} = -\nabla p_{rgh} - (\vec{g}\cdot\vec{z})\nabla\rho - \rho\vec{g} + \rho\vec{g}, \tag{2.4}$$

$$\nabla p + \rho\vec{g} = -\nabla p_{rgh} - (\vec{g}\cdot\vec{z})\nabla\rho. \tag{2.5}$$

The momentum equation is defined in `UEqn.H` file presented in List. 2.3.

Listing 2.3: `reactingFoam/rhoReactingBuoyantFoam/UEqn.H`

```
1   MRF.correctBoundaryVelocity(U);
2
3   fvVectorMatrix UEqn
4   (
5       fvm::ddt(rho, U) + fvm::div(phi, U)
6     + MRF.DDt(rho, U)
7     + turbulence->divDevRhoReff(U)
8    ==
9       fvOptions(rho, U)
10  );
11
12  UEqn.relax();
13
14  fvOptions.constrain(UEqn);
15
16  if (pimple.momentumPredictor())
17  {
18      solve
19      (
20          UEqn
21       ==
22          fvc::reconstruct
23          (
24              (
25                - ghf*fvc::snGrad(rho)
26                - fvc::snGrad(p_rgh)
27              )*mesh.magSf()
28          )
29      );
30
31      fvOptions.correct(U);
32      K = 0.5*magSqr(U);
33  }
```

The object `MRF` is related to Multiple Reference Frame, which is one method for solving problems including the rotating parts with the static mesh. More information can be found in [26]. In line 7, the `turbulence->divDevReff(U)` term is related to the turbulence RAS/LES model chosen.

The time derivative, convection and the laplacian are implicit terms stored in the `fvm` member classes in `UEqn` and it is set equal to terms in line `solve`. The `reconstruct` command reconstructs

a volume field from a face flux field. The volume field is reconstructed from face values rather than simply using the cell center values from the onset in order to create a pseudo-staggered grid setup on OpenFOAM's standard colocated grid. This method is effectively a representation of Rhie-Chow interpolation [27], which aims to remove checker-board pressure oscillations that may occur on colocated grids (due to pressure at a cell only depending on adjacent cells and not on the cell in question) [28, 29]. The two terms inside `reconstruct` are those found in the l.h.s of Eq. (2.5), where they are the negative surface normal gradient of ρ and p_{rgh} multiplied by the surface area over all of the cell faces. The resulting linear equations are then solved with a matrix solver, thus yielding the predicted velocity.

2.3 Chemical species conservation

In order to account for the chemical reactions between different chemical species, a conservation equation for each species i is given by

$$\frac{\partial \rho Y_i}{\partial t} + \nabla \cdot \rho \vec{V} Y_i = \nabla \cdot (\mu_{eff} \nabla Y_i) + \dot{\omega}_i^{'''}, \quad i = 1, .., N-1. \tag{2.6}$$

In List. 2.4, the equation that solves each species is built in line 23 of `YiEqn`, where the reacting source term, $\dot{\omega}_i^{'''}$, in Eq. (2.6) stands for `reaction->R(Yi)`. The `reactingFoam` solver was constructed to tackle, mainly, turbulent problems. Thus, as can be noted in Eq. (2.6), the transport parameter is defined as μ_{eff} as in line 27, `fvm::laplacian(turbulence->muEff(),Yi)`. Therefore, it is admited that the Schmidt number, $\mathrm{Sc} = \nu/\mathcal{D}$, is equal to unity such that the Lewis number becomes $\mathrm{Le} = 1/\mathrm{Pr}$. Using this assumption, the termo-diffusivity effect for all species becomes equal and the prefferential diffusion effects are neglected, differently than Eq. (1.9).

Listing 2.4: `reactingFoam/YEqn.H`

```
1  tmp<fv::convectionScheme<scalar>> mvConvection
2  (
3      fv::convectionScheme<scalar>::New
4      (
5          mesh,
6          fields,
7          phi,
8          mesh.divScheme("div(phi,Yi_h)")
9      )
10 );
11
12 {
13     reaction->correct();
14     Qdot = reaction->Qdot();
15     volScalarField Yt(0.0*Y[0]);
16
17     forAll(Y, i)
18     {
19         if (i != inertIndex && composition.active(i))
20         {
21             volScalarField& Yi = Y[i];
22
23             fvScalarMatrix YiEqn
24             (
25                 fvm::ddt(rho, Yi)
26               + mvConvection->fvmDiv(phi, Yi)
27               - fvm::laplacian(turbulence->muEff(), Yi)
28              ==
29                 reaction->R(Yi)
30               + fvOptions(rho, Yi)
31             );
32
33             YiEqn.relax();
34
35             fvOptions.constrain(YiEqn);
```

```
36
37              YiEqn.solve(mesh.solver("Yi"));
38
39              fvOptions.correct(Yi);
40
41              Yi.max(0.0);
42              Yt += Yi;
43          }
44      }
45
46      Y[inertIndex] = scalar(1) - Yt;
47      Y[inertIndex].max(0.0);
48  }
```

2.4 Energy conservation

In List. 2.5, the energy transport equation is based on the sensible enthalpy, `he` in OpenFOAM's nomenclature, and is defined as

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\nabla \rho \vec{V} h) + \frac{\partial \rho K}{\partial t} + \nabla \cdot (\nabla \rho \vec{V} K) - \frac{\partial p}{\partial t} = \nabla \cdot (\alpha_{eff} \nabla h) + \dot{Q}''' \tag{2.7}$$

where $K = |\vec{V}^2|/2$ is the kinetic energy, h is the enthalpy defined as the sum of internal energy e and the kinematic pressure $h = e + p/e$ [30, 24] and $\dot{Q}'''$ is the volumetric heat release rate resulting from $\dot{Q}''' = -\sum_{i=1}^{N} h_i^o \dot{\omega}_i'''$.

The effective thermal diffusivity α_{eff} is the sum of the laminar and turbulent thermal diffusivities

$$\alpha_{eff} = \frac{\rho \nu_t}{\mathrm{Pr}_t} + \frac{\mu}{\mathrm{Pr}} = \frac{\rho \nu_t}{\mathrm{Pr}_t} + \frac{\lambda}{c_p} \tag{2.8}$$

where ν_t is the turbulent (kinematic) viscosity and Pr_t is the turbulent Prandtl number. The contribution of the transient pressure term, $\partial p/\partial t$, could be present or not in the energy equation if the problem require the necessecity to solve shock waves. After solving the energy equation, in line 28, `EEqn.solve()`, the thermophysical variables are updated in line 32, `thermo.correct()`. The source codes related to the `correct()` procedure are found in `src/thermophysicalModels`.

Listing 2.5: `reactingFoam/EEqn.H`

```
1  {
2      volScalarField& he = thermo.he();
3
4      fvScalarMatrix EEqn
5      (
6          fvm::ddt(rho, he) + mvConvection->fvmDiv(phi, he)
7        + fvc::ddt(rho, K) + fvc::div(phi, K)
8        + (
9              he.name() == "e"
10           ? fvc::div
11             (
12                 fvc::absolute(phi/fvc::interpolate(rho), U),
13                 p,
14                 "div(phiv,p)"
15             )
16           : -dpdt
17         )
18       - fvm::laplacian(turbulence->alphaEff(), he)
19      ==
20         Qdot
21       + fvOptions(rho, he)
22     );
23
24     EEqn.relax();
25
```

```
26      fvOptions.constrain(EEqn);
27
28      EEqn.solve();
29
30      fvOptions.correct(he);
31
32      thermo.correct();
33
34      Info<< "min/max(T) = "
35          << min(T).value() << ", " << max(T).value() << endl;
36  }
```

2.5 Pressure equation

The pressure correction equation has the purpose of correcting the velocity field and density such that the restrictions associated with the continuity equation are satisfied [29, 31, 32]. The equation reads in the semi-discrete form

$$\frac{\partial \rho}{\partial t} V_P + \sum_{cf} \psi p \vec{V}^*_{cf} \cdot \vec{S}_{cf} + \sum_{cf} \rho^*_{cf} \frac{\mathbf{H}[\mathbf{V}^*]}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \frac{\nabla p_P}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \rho^*_{cf} \vec{V}^*_{cf} \cdot \vec{S}_{cf} + \sum_{cf} \rho^*_{cf} \frac{\mathbf{H}[\mathbf{V}']}{\mathbf{A_P}} \cdot \vec{S}_{cf} + \sum_{cf} \rho'_{cf} \vec{V}'_{cf} \cdot \vec{S}_{cf} = 0 \tag{2.9}$$

in which the subscript cf stands for cell face, P for cell center point, and S_{cf} for cell face area vector. The sum are taken over the faces around the cell with the center point, P. The asterisk stands for the temporary values computed previously in the momentum equation, which generally will not satisfy the continuity equation at the first interactions, and the superscript prime indicates the correction terms. The $\mathbf{H}[\mathbf{V}]$ operator is defined as

$$\mathbf{H}[\mathbf{V}] = \vec{r} - \sum_{nb} a^{\vec{V}}_{nb} \vec{V}_{nb} \tag{2.10}$$

The $\vec{r}$ is a source term and it is a contribution from the discretization of time-term $\frac{\partial \rho}{\partial t} V_P$ where V_P is the volume of P cell. The equation above comes from the discretization of the linearized momentum equation resulting in the system $\mathbf{A}[\mathbf{x}] = \mathbf{B}$ where a_{nb} is related to the individual contribuiton mass flux of the neighbor cells. The $\mathbf{A_P}$ is the diagonal matrix of a_P coefficients that takes into account all mass flux and time variation in the P cell.

The density is written as

$$\rho = \psi p \tag{2.11}$$

where $\psi = (R_g T)^{-1}$ is the compressibility factor. The last term in Eq. (2.9) is very small and it is neglected. The second last term is also neglected since the velocity correction $\vec{V}'$ is unknow at the moment of the solution. Hence, the final form of the pressure equation becomes

$$\frac{\partial \rho}{\partial t} V_P + \sum_{cf} \psi p \vec{V}^*_{cf} \cdot \vec{S}_{cf} + \sum_{cf} \rho^*_{cf} \frac{\mathbf{H}[\mathbf{V}^*]}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \frac{\nabla p_P}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \rho^*_{cf} \vec{V}^*_{cf} \cdot \vec{S}_{cf} = 0. \tag{2.12}$$

Then, the Eq. (2.2) for p_{rgh} is inserted in Eq. (2.12) yielding

$$\frac{\partial \rho}{\partial t} V_P + \sum_{cf} \psi (p_{rgh} + \rho \vec{g} \cdot \vec{z}) \vec{V}^*_{cf} \cdot \vec{S}_{cf} + \sum_{cf} \rho^*_{cf} \frac{\mathbf{H}[\mathbf{V}^*]}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \frac{\nabla p_{rgh,P}}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \frac{\nabla \rho \vec{g} \cdot \vec{z}}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \rho^*_{cf} \vec{V}^*_{cf} \cdot \vec{S}_{cf} = 0. \tag{2.13}$$

The above equation still contains the density ρ of the current time step. As approximation of the density of the current time step, the density of the previous time step ρ^* could be used. By doing this and with the following expression

$$\psi p_{rgh} = \psi p - \psi \rho \vec{g} \cdot \vec{z} = \rho - \psi \rho \vec{g} \cdot \vec{z} \tag{2.14}$$

the modified pressure is simplified as

$$\begin{aligned} \frac{\partial \rho}{\partial t} V_P + \sum_{cf} \psi p_{rgh} \vec{V}^*_{cf} \cdot \vec{S}_{cf} - \sum_{cf} \psi p^*_{rgh} \vec{V}^*_{cf} \cdot \vec{S}_{cf} + \sum_{cf} \rho^*_{cf} \frac{\mathbf{H}[\mathbf{V}^*]}{\mathbf{A_P}} \cdot \vec{S}_{cf} \\ - \sum_{cf} \frac{\nabla p_{rgh,P}}{\mathbf{A_P}} \cdot \vec{S}_{cf} - \sum_{cf} \frac{\nabla \rho^*_P \vec{g} \cdot \vec{z}}{\mathbf{A_P}} = 0. \end{aligned} \tag{2.15}$$

Comparing the above equation with the source code in List. 2.6 we can identify the corrected phase velocity without considering the pressure gradient as

$$\vec{V}^*_{cf} = \frac{\mathbf{H}[\mathbf{V^*}]}{\mathbf{A_P}} - \frac{\nabla \rho^*_P \vec{g} \cdot \vec{z}}{\mathbf{A_P}}. \tag{2.16}$$

In order to derive the expression for the time derivative in the source code of the pressure equation, the density is divided into the density of the previous time step ρ^* and a density correction $\rho^{'}$, i.e., $\rho = \rho^* + \rho^{'}$ and the time derivative is taken from this expression

$$\frac{\partial \rho}{\partial t} = \frac{\partial \rho^*}{\partial t} + \frac{\partial \rho^{'}}{\partial t} = \frac{\partial \rho^*}{\partial t} + \psi \left(\frac{\partial p^{'}_{rgh}}{\partial t} + \frac{\partial \rho^{'}}{\partial t} \vec{g} \cdot \vec{z} \right). \tag{2.17}$$

Neglecting the last two terms in the above equation, it yields

$$\frac{\partial \rho}{\partial t} = \frac{\partial \rho^*}{\partial t} + \psi \frac{\partial p^{'}_{rgh}}{\partial t} \tag{2.18}$$

Listing 2.6: `reactingFoam/rhoReactingBuoyantFoam/pEqn.H`

```
rho = thermo.rho();

// Thermodynamic density needs to be updated by psi*d(p) after the
// pressure solution
const volScalarField psip0(psi*p);

volScalarField rAU(1.0/UEqn.A());
surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));
volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));

surfaceScalarField phig(-rhorAUf*ghf*fvc::snGrad(rho)*mesh.magSf());

surfaceScalarField phiHbyA
(
    "phiHbyA",
    (
        fvc::flux(rho*HbyA)
      + MRF.zeroFilter(rhorAUf*fvc::ddtCorr(rho, U, phi))
    )
  + phig
);

MRF.makeRelative(fvc::interpolate(rho), phiHbyA);

// Update the pressure BCs to ensure flux consistency
constrainPressure(p_rgh, rho, U, phiHbyA, rhorAUf, MRF);

fvScalarMatrix p_rghDDtEqn
```

```
(
    fvc::ddt(rho) + psi*correction(fvm::ddt(p_rgh))
  + fvc::div(phiHbyA)
  ==
    fvOptions(psi, p_rgh, rho.name())
);

while (pimple.correctNonOrthogonal())
{
    fvScalarMatrix p_rghEqn
    (
        p_rghDDtEqn
      - fvm::laplacian(rhorAUf, p_rgh)
    );

    p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.finalInnerIter())));

    if (pimple.finalNonOrthogonalIter())
    {
        // Calculate the conservative fluxes
        phi = phiHbyA + p_rghEqn.flux();

        // Explicitly relax pressure for momentum corrector
        p_rgh.relax();

        // Correct the momentum source with the pressure gradient flux
        // calculated from the relaxed pressure
        U = HbyA + rAU*fvc::reconstruct((phig + p_rghEqn.flux())/rhorAUf);
        U.correctBoundaryConditions();
        fvOptions.correct(U);
        K = 0.5*magSqr(U);
    }
}

p = p_rgh + rho*gh;

// Thermodynamic density update
thermo.correctRho(psi*p - psip0);

if (thermo.dpdt())
{
    dpdt = fvc::ddt(p);
}

#include "rhoEqn.H"
#include "compressibleContinuityErrs.H"
```

Chapter 3

FGM solver

In this implementation, the FGM solver, `fgmPremixedFoam`, is built from the `rhoReactingBuoyantFoam` where a new library named `combustionFGMModel` is called by the OpenFOAM. A general view of this implementation is seen in Fig. 3.1. The `fgmPremixedFoam` solver resolves the mass, momentum and progress variable balance equations. The new library has two main classes, `lookupFGM` and `fgmThermo`, responsible for computing the thermochemistry fields and returning them to the main solver. The manifold containing the thermo-chemistry data is loaded by the `lookupFGM` class, which is held in the `constant/` directory of the case to be solved. In the following discussion, we present the top-level changes in `rhoReactingBuoyantFoam` and the new library details.

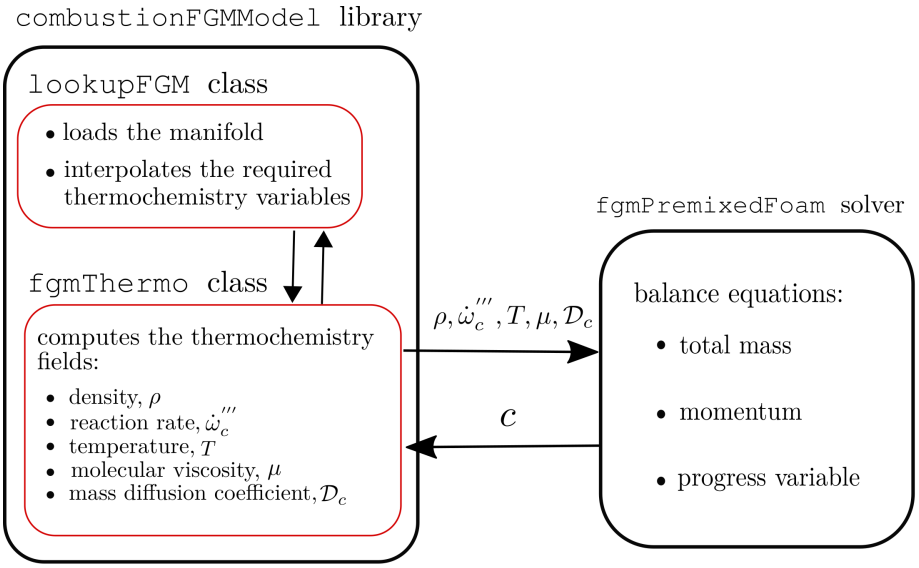


Figure 3.1: Schematic workflow of the `fgmPremixedFoam` solver and the `combustionFGMModel` library.

3.1 Main solver changes

In `rhoReactingBuoyantFoam` solver, the thermo object is declared in `createFields.H` as seen in List. 3.1.

Listing 3.1: `reactingFoam/rhoReactingBuoyantFoam/createFields.H`

```
3 Info<< "Reading thermophysical properties\n" << endl;
4 autoPtr<rhoReactionThermo> pThermo(rhoReactionThermo::New(mesh));
5 rhoReactionThermo& thermo = pThermo();
6 thermo.validate(args.executable(), "h", "e");
7
8 basicSpecieMixture& composition = thermo.composition();
```

In this solver, the lines above are replaced by the following code observed in List. 3.2. The progress variable, c, is declared as `PV` in line 4, and the pressure field, p, is declared in line 17. The objects related to classes `lookupFGM` and `fgmThermo` are initialized in lines 32 and 35, respectively.

Listing 3.2: `fgmPremixedFoam/createFields.H`

```
3  // FGM Fields
4  volScalarField PV
5  (
6      IOobject
7      (
8          "PV",
9          runTime.timeName(),
10         mesh,
11     IOobject::MUST_READ,
12     IOobject::AUTO_WRITE
13     ),
14     mesh
15 );
16
17 volScalarField p
18 (
19     IOobject
20     (
21         "p",
22         runTime.timeName(),
23         mesh,
24     IOobject::MUST_READ,
25     IOobject::AUTO_WRITE
26     ),
27     mesh
28 );
29
30 Info<< "Reading thermophysical properties\n" << endl;
31 // Initializing FGM manifold
32 lookupFGM fgmTable(mesh);
33
34 // Initializing fgmThermo
35 fgmThermo thermo(mesh, fgmTable, p, PV);
```

As pointed out in Chapter 1, the FGM model simplifies the species and the energy equation. Hence, lines 107 and 108 which are default in `rhoReactingBuoyantFoam.C`, List. 2.1, are replaced by the solution of `PVEqn.H` in `fgmPremixedFoam.C`, List. 3.3.

Listing 3.3: `fgmPremixedFoam/fgmPremixedFoam.C`

```
94 #include "UEqn.H"
95 #include "PVEqn.H"
```

The progress variable `PV` is solved as a transport equation as seen in List. 3.4. The thermophysical variables `thermo.Dmass()` and `thermo.sourcePV()` are computed in the `fgmThermo` class. In addition, the temperature field also comes out from `thermo` object and all transport and thermodynamic properties are updated in line 33, `thermo.correct()`.

Listing 3.4: `fgmPremixedFoam/PVEqn.H`

```
tmp<fvScalarMatrix> tPVEqn
    (
    (
      fvm::ddt(rho, PV)
    + fvm::div(phi, PV)
      - fvm::laplacian(thermo.Dmass(), PV)
      == thermo.sourcePV()
        )
    );

    fvScalarMatrix& PVEqn = tPVEqn.ref();

    PVEqn.relax();
    fvOptions.constrain(PVEqn);

    PVEqn.solve();
    fvOptions.correct(PV);
    PV = max(min( PV, 1.0 ), 0.0 );

    T = thermo.T();

    thermo.correct();
```

The `fgmPremixedFoam` solver aims to model problems within the scope of DNS, therefore, no turbulence modeling is used and the momentum equation is set up accordingly, List. 3.5.

Listing 3.5: `fgmPremixedFoam/UEqn.H`

```
    fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U) + fvm::div(phi, U) + MRF.DDt(rho, U)
      - fvm::laplacian(thermo.mu(), U)
      - fvc::div(thermo.mu()*Foam::dev2(Foam::T(fvc::grad(U))))
     ==
      fvOptions(rho, U)
    );
```

3.2 Thermophysical library

The `combustionFGMModel` is made up of the `lookupFGM` and `fgmThermo` classes. The `fgmTable` object is initialized in `createFields.H` as a `lookupFGM` class and loads the manifolds in the `constant` folder, List. 3.6, and also, it is responsible for interpolating the required thermochemistry variables, ρ, T, $\dot{\omega}_c'''$, in order to obtain the local values of the progress variable in `PVEqn.H`.

Listing 3.6: `combustionFGMModel/lookupFGM.C`

```
lookupFGM::lookupFGM
(
 const fvMesh& mesh
)
  :
  IOdictionary
  (
    IOobject
    (
     "fgmProperties",
     mesh.time().constant(),
     mesh,
     IOobject::MUST_READ_IF_MODIFIED,
     IOobject::NO_WRITE
     )
  ),
  mesh_(mesh),
```

```
  PV_table(lookup("PV")),
  sourcePV_table(lookup("sourcePV") ),
  T_table(lookup("T") ),
  rho_table(lookup("rho"))

...
```

The `lookupFGM` member function, `lookupFGM::interpolateValue1D`, computes the interpolation process of ρ, T and the source term, $\dot{\omega}_c'''$, List. 3.7, as the progress variable c/`PV` is solved in `PVEqn.H` and then, the transport and thermodynamic properties are updated.

Listing 3.7: `combustionFGMModel/lookupFGM.C`

```
Foam::scalar Foam::lookupFGM::interpolateValue1D
(
    const List<scalar>& table,
    scalar pvValue,
    const List<scalar>& pvTable
) const
{

    scalar interpolatedValue;

    scalar lower_pvTable=0;
    scalar upper_pvTable=0;
    scalar lower_table=0;
    scalar upper_table=0;

     if(pvValue == 0)
       {
     interpolatedValue = table[0];
       }
     else
       {
     //- A small number to prevent divide by zero
         scalar smallValue(1e-5);
         scalar rate;

        // INTERPOLATION ALGORITHM
        for(int j=0; j < pvTable.size(); j++ )
         {
           pvValue = min(pvValue,1.);

        if(pvTable[j] >= pvValue)
     {

        lower_pvTable = pvTable[j-1];
            upper_pvTable = pvTable[j];

        lower_table = table[j-1];
            upper_table = table[j];

        break;
          }

         }

     rate = (upper_table - lower_table )/ \
         max((upper_pvTable - lower_pvTable),smallValue);

         interpolatedValue = ( pvValue - lower_pvTable )*rate + lower_table;

         interpolatedValue = max(interpolatedValue,min(table));

        }

    return interpolatedValue;
```

```
128
129 }
```

The `fgmThermo` class is concerned with computing all thermodynamic states and the molecular transport variables, e.g., dynamic viscosity, μ, and the mass diffusivity coefficient, $\mathcal{D}$, List. 3.8. This class was built according to the OpenFOAM's library `$FOAM_SRC/thermophysicalModels/`. The data file in the `constant` folder has the information about the FGM model, `thermoType`, the Lewis number and the gas constant R_{gas}. The `fgmTable_` is loaded from the `lookupFGM` class and the variables `p_` and `PV_` come from the initialization in `createFields.H`. The other members data are also initialized and can be checked in the files provided with this report.

Listing 3.8: Piece of initialization variables in `combustionFGMModel/fgmThermo/fgmThermo.C`

```
74 Foam::fgmThermo::fgmThermo
75 (
76     const fvMesh& mesh,
77     const lookupFGM& lookupFGM,
78     volScalarField& p,
79     volScalarField& PV
80 )
81 :
82
83   IOdictionary
84   (
85     IOobject
86     (
87      "thermophysicalProperties",
88      mesh.time().constant(),
89      mesh,
90      IOobject::MUST_READ_IF_MODIFIED,
91      IOobject::NO_WRITE
92      )
93   ),
94
95   fgmThermoModel_(lookup("thermoType")),
96   Le_(lookupOrDefault<scalar>("Le",1.0)),
97   Rgas_(lookupOrDefault<scalar>("Rgas",287.0)),
98
99   fgmTable_(lookupFGM),
100
101  p_(p),
102
103  PV_(PV),
104
105 ...
```

The `Foam::fgmThermo::correct()` member function is responsible for updating all variables as the transport equation for `PV` is solved. A small section of `correct()` is shown in List. 3.9.

Listing 3.9: Piece of interpolation process in `combustionFGMModel/fgmThermo/fgmThermo.C`

```
175 void Foam::fgmThermo::correct()
176 {
177   scalarField& TCells = T_.primitiveFieldRef();
178   scalarField& psiCells = psi_.primitiveFieldRef();
179   scalarField& sourcePVCells = sourcePV_.primitiveFieldRef();
180   scalarField& rhoCells = rho_.primitiveFieldRef();
181   scalarField& DmassCells = Dmass_.primitiveFieldRef();
182   scalarField& muCells = mu_.primitiveFieldRef();
183
184   const scalarField& pCells = p_.internalField();
185   const scalarField& PVCells = PV_.internalField();
186
187   // Interpolate for internal field
188   forAll(TCells, celli)
```

```
189   {
190       TCells[celli] = fgmTable_.interpolateValue1D
191                          (
192                       fgmTable_.T_table,
193                       PVCells[celli],
194                       fgmTable_.PV_table
195                          );
196
197       DmassCells[celli] = massDiffusivity_model(TCells[celli]);
198
199       muCells[celli] = viscosity_model(TCells[celli]);
200
201 ...
```

The values of members data, `T_`, `rho_`, and `sourcePV_` are corrected for each cell using the interpolation method. The transport properties, `Dmass_` and `mu_`, are calculated following the member functions implemented according to Eq. (1.23), for both mass diffusivity and the Sutherland's law for viscosity, List. 3.10. The ideal gas law, $\psi = \rho/p$ is used for `psi_`. The members data are also updated for the patches.

Listing 3.10: Member functions to compute the transport variables in `fgmThermo.C`

```
285 Foam::scalar Foam::fgmThermo::compressibility_model(const scalar rho, const scalar p) const
286 {
287   return rho/p;
288 }
289
290 Foam::scalar Foam::fgmThermo::massDiffusivity_model(const scalar T) const
291 {
292
293   scalar T298 = 298;
294   scalar C069 = 0.69;
295   scalar CD = 2.58E-5;
296
297   return CD*pow(T/T298,C069)/Le_;
298
299 }
300
301 Foam::scalar Foam::fgmThermo::viscosity_model(const scalar T) const
302 {
303   // Model: Sutherland's law
304   scalar muRef = 1.7894E-5;
305   scalar TRef = 273.15;
306   scalar S = 110.4;
307
308   return muRef*pow(T/TRef,1.5)*((TRef+S)/(T+S));
309
310 }
```

3.3 Solver compilation

The solver compilation can be performed by running the provided bash script in the `fgmPremixedFoam/` folder

```
bash Allwmake
```

or can be done manually typing in terminal window

```
wmakeLnInclude -u combustionFGMModel
wmake combustionFGMModel
wmake fgmPremixedFoam
```

Chapter 4

2D Bunsen flame test case

In this Chapter, it is built a test case for a two-dimensional slab domain subject to a prescribed inlet velocity, while at the sides of the domain, symmetric boundary conditions were imposed, Fig. 4.1. The domain has a length of $L = 20$ mm in the streamwise direction and a width of $W = 5$ mm. To avoid reflections of pressure waves at the inlet and at the outlet region, partially non-reflective boundary conditions (PNRBC) are used [33]. A constant Poiseuille flow is prescribed at the inlet domain. The simulation starts by initializing a value of the progress variable equal to $c(t = 0) = 0.8$ in the entire domain. The setup is done using the tutorial case from the `buoyantPimpleFoam/hotRoom/` problem.

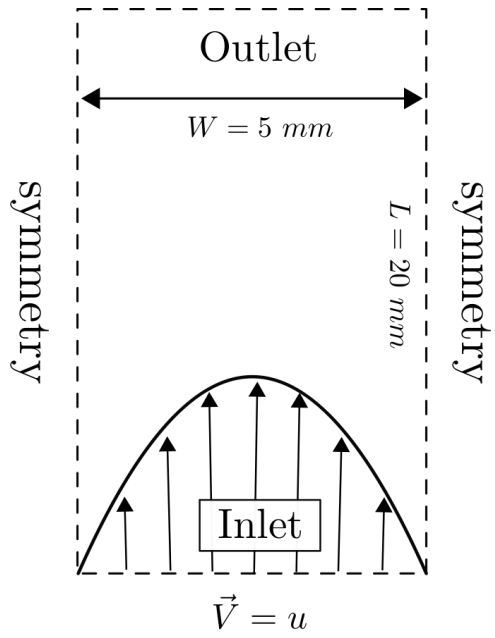


Figure 4.1: Initial and boundary conditions for the Bunsen flame test case.

4.1 Geometry

Copy the `hotRoom/` from OpenFOAM's tutorial to the local folder `$FOAM_USER/`.

```
cd $WM_PROJECT_USER_DIR
mkdir testCase
cp -r $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/hotRoom ./testCase
```

```
cd testCase
mv hotRoom bunsenFlame
cd bunsenFlame
```

Now, the geometry and mesh are setup. First, the domain dimensions are configured.

```
sed -i s/"scale \ \ 1"/"scale \ \ 0.001"/g system/blockMeshDict
sed -i s/"10 0 0"/"20 0 0"/g system/blockMeshDict
sed -i s/"10 5 0"/"20 5 0"/g system/blockMeshDict
sed -i s/"(0 0 10)"/"(0 0 0.1)"/g system/blockMeshDict
sed -i s/"10 0 10"/"20 0 0.1"/g system/blockMeshDict
sed -i s/"10 5 10"/"20 5 0.1"/g system/blockMeshDict
sed -i s/"0 5 10"/"0 5 0.1"/g system/blockMeshDict
```

Then, the mesh resolution is set.

```
sed -i s/"(20 10 20)"/"(600 100 1)"/g system/blockMeshDict
```

The patches names and locations are defined.

```
sed -i s/floor/inlet/g system/blockMeshDict
sed -i s/"(1 5 4 0)"/"(0 4 7 3)"/g system/blockMeshDict
sed -i s/ceiling/outlet/g system/blockMeshDict
sed -i s/"(3 7 6 2)"/"(1 2 6 5)"/g system/blockMeshDict
sed -i s/"type wall;"/"type patch;"/g system/blockMeshDict
sed -i "$(sed -n '/patch/ =' system/blockMeshDict | tail -n 1)"' s/patch/empty/' system/blockMeshDict
sed -i s/fixedWalls/frontAndBack/g system/blockMeshDict
sed -i '/\ \ \ frontAndBack/i \ \ \ lowerWall\n\ \ \ {\n\ \ \ }' system/blockMeshDict
sed -i '/\ \ \ lowerWall/i \ \ \ upperWall\n\ \ \ {\n\ \ \ }' system/blockMeshDict
sed -i "$(sed -n '/(0 4 7 3)/ =' system/blockMeshDict | tail -n 1)"' s/(0 4 7 3)//' \
system/blockMeshDict
sed -i "$(sed -n '/(2 6 5 1)/ =' system/blockMeshDict | head -n 1)"' s/(2 6 5 1)//' \
system/blockMeshDict
```

From the former lines, we created two new patches. Copy the lines below and put them within the curly brackets of `upperWall{}` and `lowerWall{}`, respectively, found in the `system/blockMeshDict`.

`upperWall` patch

```
type symmetry;
faces
(
  (7 6 2 3)
);
```

`lowerWall` patch

```
type symmetry;
faces
(
  (0 1 5 4)
);
```

4.2 Boundary conditions

The vector and the scalar fields are configured in the `0/` folder. The original turbulent fields `alphat,epsilon,k,nut` and the temperature field are removed. The same process done before is carried out for `p,p_rgh,U`.

```
mv 0.orig 0
cd 0
rm alphat epsilon k nut T
sed -i s/floor/inlet/g p
sed -i s/floor/inlet/g p_rgh
sed -i s/floor/inlet/g U
sed -i s/ceiling/outlet/g p
```

```
sed -i s/ceiling/outlet/g p_rgh
sed -i s/ceiling/outlet/g U
sed -i s/fixedWalls/frontAndBack/g p
sed -i s/fixedWalls/frontAndBack/g p_rgh
sed -i s/fixedWalls/frontAndBack/g U
sed -i "/\ \ \ \ frontAndBack/i \ \ \ lowerWall\n\ \ \ {\n\ \ \ }" p
sed -i "/\ \ \ \ frontAndBack/i \ \ \ lowerWall\n\ \ \ {\n\ \ \ }" p_rgh
sed -i "/\ \ \ \ frontAndBack/i \ \ \ lowerWall\n\ \ \ {\n\ \ \ }" U
sed -i "/\ \ \ lowerWall/i \ \ \ upperWall\n\ \ \ {\n\ \ \ }" p
sed -i "/\ \ \ lowerWall/i \ \ \ upperWall\n\ \ \ {\n\ \ \ }" p_rgh
sed -i "/\ \ \ lowerWall/i \ \ \ upperWall\n\ \ \ {\n\ \ \ }" U
sed -i "$(sed -n '/calculated;/ =' p | tail -n 1)"' s/calculated;/empty;/' p
sed -i "$(sed -n '/fixedFluxPressure;/ =' p_rgh | tail -n 1)"' s/fixedFluxPressure;/empty;/' p_rgh
sed -i "$(sed -n '/noSlip;/ =' U | tail -n 1)"' s/noSlip;/empty;/' U
sed -i "$(sed -n '/value \ \ \ \ \ \ \ \ \ \ $internalField;/ =' p | tail -n 1)"\
' s/value \ \ \ \ \ \ \ \ \ \ $internalField;//' p
sed -i "$(sed -n '/value \ \ \ \ \ \ \ \ \ \ uniform 1e5;/ =' p_rgh | tail -n 1)"\
' s/value \ \ \ \ \ \ \ \ \ \ uniform 1e5;//' p_rgh
```

For all fields, the boundary conditions are set up as symmetric on the domain sides, for `upperWall` and `lowerWall` patches.

Sides boundary condition for `upperWall` and `lowerWall` patches

```
type symmetry;
```

In the `p_rgh` file, copy the wave transmissive boundary condition for the outlet patch.

`p_rgh` outlet boundary condition

```
type         waveTransmissive;
value        $internalField;
field        p;
gamma        1.3;
fieldInf     1e5;
lInf         0.1;
```

In the `U` file, copy the following boundary condition for the outlet patch.

`U` outlet boundary condition

```
type            inletOutlet;
inletValue      uniform (0 0 0);
value           uniform (0 0 0);
```

A Poiseulle flow velocity will be imposed as the inlet boundary condition at `U`. First, the inlet boundary condition is modified.

```
sed -i ' s/\ \ \ \ \ \ \ type \ \ \ \ \ \ \ \ \ \ noSlip;/ \
\ \ \ \ \ \ \ \#include inletPoiseulleFlow;/' U
touch inletPoiseulleFlow
```

Then, paste the following code in the `inletPoiseulleFlow` file.

`bunsenFlame/0/inletPoiseulleFlow`

```
    type            codedFixedValue;
    value           uniform (0 0 0);
    name            codedStuff;
    codeInclude
    #{
    #};

    code
    #{
```

```
    const fvPatch& boundaryPatch = patch();
    const vectorField& Cf = boundaryPatch.Cf();
    vectorField& field = *this;

    const scalar r = 0.0025;
    const scalar Sl_o =0.371;
    const scalar uMean = 2*Sl_o;

    // Poiseulle Flow
    forAll(Cf, faceI)
    {
      const scalar y = Cf[faceI].y(); // y coordinate
      const scalar u = 1.5*uMean*(1 - (pow((y - r)/r, 2)));

      field[faceI] = vector(u, 0, 0);
    }
       #};
```

The progress variable PV is created from the p field. As already mentioned, the initial condition is set up as PV=0.8 in the entire domain.

```
cp p PV
sed -i s/"object \ \ \ \ \ p"/"object \ \ \ \ \ PV"/g PV
sed -i s/"1 -1 -2 0 0 0 0"/"0 0 0 0 0 0 0"/g PV
sed -i s/1e5/0.8/g PV
sed -i "$(sed -n '/calculated/ =' PV | head -n 1)"' s/calculated/fixedValue/' PV
sed -i "$(sed -n '/$internalField/ =' PV | head -n 1)"' s/$internalField/uniform 0/' PV
sed -i "$(sed -n '/calculated/ =' PV | head -n 1)"' s/calculated/zeroGradient/' PV
sed -i "$(sed -n '/value \ \ \ \ \ \ \ \ \ \ $internalField;/ =' PV | head -n 1)"\
' s/value \ \ \ \ \ \ \ \ \ \ $internalField;//' PV
```

4.3 Thermophysical properties

The thermophysical properties are defined in constant/thermophysicalProperties. Here, for the fgmPremixedFoam

```
cd ..
sed -i '17,46d' constant/thermophysicalProperties
```

and copy and paste the piece of code in thermophysicalProperties.

bunsenFlame/constant/thermophysicalProperties

```
thermoType fgm1DModelDNS;

// Lewis number of CO2
Le 1.384;

// Ideal gas mixture constant [J/Kg K]
Rgas 287.05;

// Pressure-work
dpdt false;
```

In this case, the gravitational effect is neglected.

```
rm constant/turbulenceProperties
sed -i s/"0 -9.81 0"/"0 0 0"/g constant/g
```

4.4 Solution and schemes

The files controlDict,fvSchemes,fvSolution are configurated in the system/ folder.

```
rm system/setFieldsDict
sed -i s/"deltaT \ \ \ \ \ \ \ \ 2;"/"deltaT \ \ \ \ \ \ \ \ 1e-15;"/g system/controlDict
sed -i s/"writeControl \ \ \ timeStep;"/"writeControl \ \ \ adjustableRunTime;"/g system/controlDict
sed -i s/"writeInterval \ \ 100;"/"writeInterval \ \ 5e-5;"/g system/controlDict
sed -i s/"endTime \ \ \ \ \ \ \ 2000;"/"endTime \ \ \ \ \ \ \ 0.01;"/g system/controlDict
sed -i s/"adjustTimeStep \ no;"/"adjustTimeStep \ yes;"/g system/controlDict
sed -i s/"maxCo \ \ \ \ \ \ \ \ 0.5;"/"maxCo \ \ \ \ \ \ \ \ 0.2;"/g system/controlDict
sed -i '51,64d' system/controlDict
```

In the `fvSchemes`, the lines between the `divSchemes{...}` should be replaced by

system/fvSchemes

```
div(phi,U)          Gauss limitedLinearV 1;
div(phi,PV)         Gauss limitedLinear 1;
div((thermo:mu*dev2(T(grad(U)))))    Gauss linear;
div((mu*dev2(T(grad(U)))))           Gauss linear;
```

At the end of the file, add the following lines after the `snGradSchemes` setup

system/fvSchemes

```
fluxRequired
{
    default yes;
    p;
}
```

For the numerical solution in `fvSolution`

```
sed -i s/1e-6/1e-8/g system/fvSolution
sed -i s/0.1/0.01/g system/fvSolution
sed -i s/"(U|h|e|k|epsilon|R)"/"(U|PV)"/g system/fvSolution
sed -i s/"(U|h|e|k|epsilon|R)Final"/"(U|PV)Final"/g system/fvSolution
sed -i s/"pRefCell \ \ \ \ \ \ \ 0;"/""/g system/fvSolution
sed -i s/"pRefValue \ \ \ \ \ \ 1e5;"/""/g system/fvSolution
```

and add the solution criteria for `p` in the `fvSolution`. The lines below should be copy and paste inside `solvers{...}` setup

system/fvSolution

```
p
{
 solver           PCG;
 preconditioner   DIC;
 tolerance        1e-08;
 relTol           1e-2;
}

pFinal
{
 $p;
 relTol 0;
}
```

4.5 Running the case

Before start the simulation, allocate the manifold `fgmProperties` file into `constant/` directory. Then, to run the case just type in the terminal window the following command

```
blockMesh
fgmPremixedFoam >& log&
```

The simulation can take more than one hour depending on the computational architecture available. Thus, in order to speed up, the simulation can be run in parallel. Thereby, copy and paste the tutorial case from `simpleFoam/pipeCyclic/` where the processes decomposition use the `scotch` method.

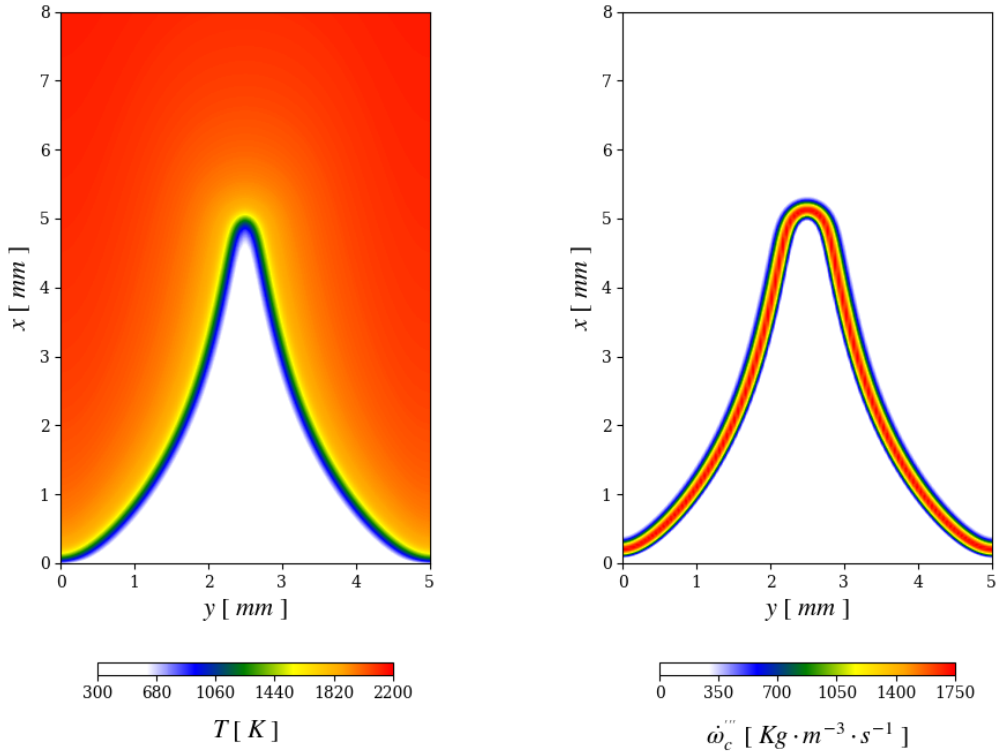
```
cp $FOAM_TUTORIALS/incompressible/simpleFoam/pipeCyclic/system/decomposeParDict system/
sed -i s/5/4/g system/decomposeParDict

decomposePar
mpirun -np 4 fgmPremixedFoam -parallel >& log&
```

After the simulation has finished, type in the terminal window to reconstruct the time steps

```
reconstructPar
```

The steady-state solution of the proposed test case should result in the contours according to Fig. 4.2.



(a) Temperature field.

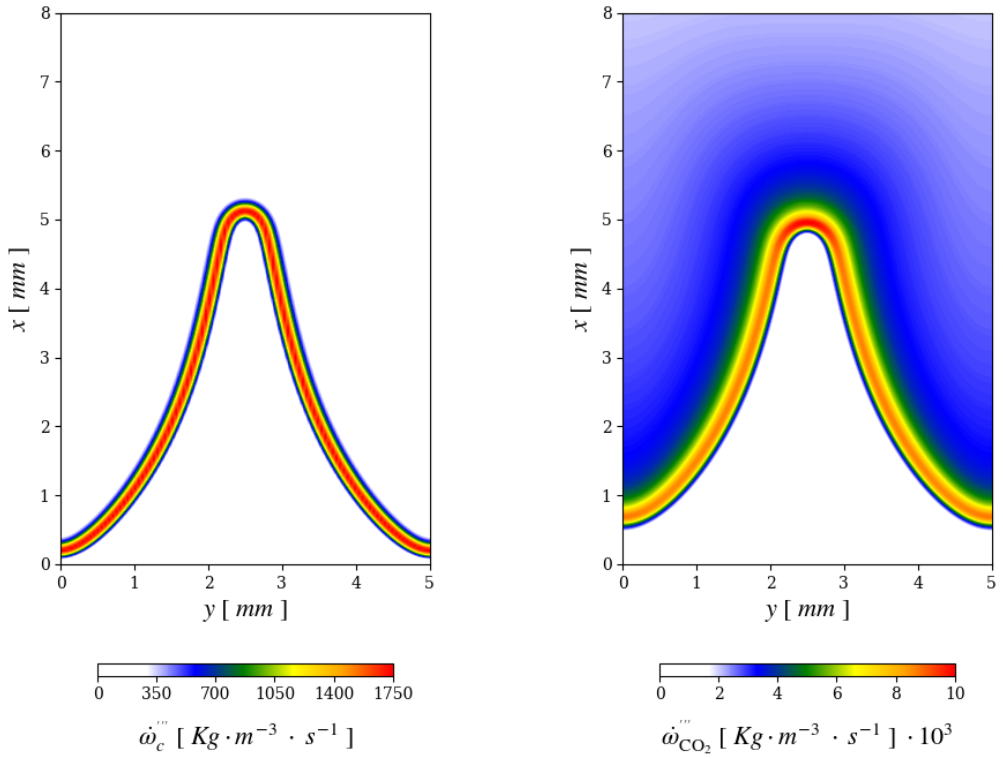
(b) Reaction rate for the progress variable c.

Figure 4.2: 2D Bunsen flame steady-state solution.

Chapter 5

Results and discussions

Figure 5.1 compares the results of the FGM model with the same case performed in the EBI-DNS (Engler-Bunte-Institute) code [16]. With EBI-DNS, the fully compressible Navier-Stokes, species and energy equations for reacting gas mixtures are solved coupled to the chemical kinetics library Cantera [34]. The reduced mechanism of Kee *et al.* [35] with 17 species and 58 elementary reactions was used to run the EBI-DNS case. In Figs. 5.1 can be seen the comparison of the production rate of variable c, Fig. 5.1a, with the production rate of CO_2 in the detailed case, Fig. 5.1b.



(a) `fgmPremixedFoam`: reaction rate for the progress variable c.

(b) EBI-DNS: reaction rate for CO_2.

Figure 5.1: Comparison of steady-state solution between the solvers `fgmPremixedFoam` and EBI-DNS.

The tips of both flames reach the same high with a difference in the lift-off effect. Figure 5.1b

shows that the flame base stabilizes farther from the domain inlet than it is seen in Fig. 5.1a. It occurs because, at the base, the local flame speed of Fig. 5.1b is lower than the reference flame speed, S_l^0. At the same time, the local flame speed in the tip of Fig. 5.1b is higher than S_l^0. These variations can be observed along the flame front, where at the flame base $\dot{\omega}'''_{CO_2} = 8 \cdot 10^3$ Kg$\cdot$m$^{-3}\cdot$s^{-1} and it varies until reach $\dot{\omega}'''_{CO_2} = 10 \cdot 10^3$ Kg$\cdot$m$^{-3}\cdot$s^{-1} at the flame tip. The changes observed in the CO_2 production rate are caused by the effects of the front stretch rate (curvature and strain rate) coupled with thermo-diffusivity instabilities [8]. Since, for the present model, the detailed effects of thermo-diffusivity instabilities are neglected, the flame surface has a constant production rate of c regardless of the flame geometry. Therefore, the local flame speed keeps constant and equal to the S_l^0 along the front. Nevertheless, the results showed in Fig. 5.1 prove that the model is able to generate similar behavior when compared to a detailed model.

In terms of performance, it is noteworthy that the case of Fig. 5.1b was run in a HPC structure with 20 cores in parallel, taking 2 days to reach the steady-state solution. On the other hand, the `fgmPremixedFoam` takes around 1 hour on a personal laptop with 4 cores running in parallel.

Bibliography

[1] N. Peters, *Reducing mechanisms*, pp. 48–67. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991.

[2] U. Maas and S. Pope, "Simplifying chemical kinetics: Intrinsic low-dimensional manifolds in composition space," *Combustion and Flame*, vol. 88, no. 3, pp. 239–264, 1992.

[3] S. Lam and D. Goussis, "Understanding complex chemical kinetics with computational singular perturbation," *Symposium (International) on Combustion*, vol. 22, no. 1, pp. 931–941, 1989.

[4] J. A. van Oijen and L. de Goey, "Modelling of premixed laminar flames using flamelet-generated manifolds," *Combustion Science and Technology*, vol. 161, no. 1, pp. 113–137, 2000.

[5] J. van Oijen, A. Donini, R. Bastiaans, J. ten Thije Boonkkamp, and L. de Goey, "State-of-the-art in premixed combustion modeling using flamelet generated manifolds," *Progress in Energy and Combustion Science*, vol. 57, pp. 30–74, 2016.

[6] R. B. Meier and A. A. Oliveira, "Detailed numerical simulation of the flame-flow interaction in a slot-burner laminar premixed flame," *In: Proceedings of 18th Brazilian Congress of Thermal Sciences and Engineering*, 2020.

[7] N. Peters, *Turbulent Combustion*. United Kingdom: Cambridge University Press, 2000.

[8] T. Poinsot and D. Veynante, *Combustion Theory: The Fundamental Theory of Chemically Reacting Flow Systems*. Philadelphia, USA: Edwards, 2005.

[9] C. Law, *Combustion Physics*. Germany: Princeton University, 2006.

[10] G. K. Giannakopoulos, A. Gatzoulis, C. E. Frouzakis, M. Matalon, and A. G. Tomboulides, "Consistent definitions of "flame displacement speed" and "markstein length" for premixed flame propagation," *Combustion and Flame*, vol. 162, no. 4, pp. 1249–1264, 2015.

[11] N. Peters, "Laminar flamelet concepts in turbulent combustion," *Symposium (International) on Combustion*, vol. 21, no. 1, pp. 1231–1250, 1988.

[12] T. C. de Souza, *Modulated Turbulence for Premixed Flames*. PhD thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 2014.

[13] J. A. M. de Swart, R. J. M. Bastiaans, J. A. van Oijen, L. P. H. de Goey, and R. S. Cant, "Inclusion of preferential diffusion in simulations of premixed combustion of hydrogen/methane mixtures with flamelet generated manifolds," *Flow, Turbulence and Combustion*, vol. 85, pp. 473–511, 2010.

[14] A. C. Contini, L. S. Donatti, C. A. Hoerlle, L. Zimmer, and F. Pereira, "Numerical study of the laminar premixed flame stabilization on a slot burner: comparison between detailed and fgm models," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 42, no. 189, 2020.

[15] J. A. van Oijen, *Flamelet-generated manifolds: development and application to premixed laminar flames*. PhD thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 2002.

[16] T. Zirwes, F. Zhang, P. Habisreuther, M. Hansinger, H. Bockhorn, M. Pfitzner, and D. Trimisn, "Quasi-dns dataset of a piloted flame with inhomogeneous inlet conditions," *Flow, Turbulence and Combustion*, vol. 104, pp. 997–1027, 2020.

[17] T. T. Zirwes, *Memory Effects in Premixed Flames:Unraveling Transient Flame Dynamics with the Flame Particle Tracking Method*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2021.

[18] R. Kee, M. Coltrin, and P. Glarborg., *Chemically Reacting Flow: Theory and Practice*. 2005.

[19] T. C. de Souza, R. Bastiaans, L. D. Goey, and B. Geurts, "Modulation of a methane bunsen flame by upstream perturbations," *Journal of Turbulence*, vol. 18, no. 4, pp. 316–337, 2017.

[20] C. J. Ball and P. M. Gill, "Chem1d: a software package for electronic structure calculations on one-dimensional systems," *Molecular Physics*, vol. 113, no. 13-14, pp. 1843–1857, 2015.

[21] G. P. Smith, D. M. Golden, M. Frenklach, N. W. Moriarty, B. Eiteneer, M. Goldenberg, C. T. Bowman, R. K. Hanson, S. Song, W. C. G. Jr, V. V. Lissianski, and Z. Quin, "Grimech 3.0 reaction mechanism," *Sandia National Laboratory*, 2000.

[22] A. Vreman, J. van Oijen, L. de Goey, and R. Bastiaans, "Subgrid scale modeling in large-eddy simulation of turbulent combustion using premixed flamelet chemistry," *Flow, Turbulence and Combustion*, vol. 82, pp. 511–535, 2009.

[23] A. Vreman, R. Bastiaans, and B. J. Geurts, "Similarity subgrid model for premixed turbulent combustion," *Flow, Turbulence and Combustion*, vol. 82, pp. 233–248, 2009.

[24] "doc.cfd.direct." https://doc.cfd.direct/notes/cfd-general-principles/. Accessed: 2022-12-8.

[25] "openfoamwiki." https://openfoamwiki.net/index.php/BuoyantBoussinesqPisoFoam. Accessed: 2022-12-6.

[26] "openfoamwiki." https://openfoamwiki.net/index.php/ChtMultiRegionFoam. Accessed: 2022-12-5.

[27] C. Rhie and W. Chow, "A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation," *AIAA Journal*, vol. 21, p. 1525–1532, 1983.

[28] T. Uroić, *Implicitly Coupled Finite Volume Algorithms*. PhD thesis, University of Zagreb, Zagreb, Croatia, 2019.

[29] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*. Berlin: Springer, 2002.

[30] "caefn." https://caefn.com/openfoam/solvers-buoyantpimplefoam. Accessed: 2022-12-8.

[31] F. Moukalled and M. Darwish, "A unified formulation of the segregated class of algorithms for multi-fluid flow at all-speeds," *Numerical Heat Transfer Part B-fundamentals*, vol. 37, p. 92, 11 1999.

[32] M. D. F. Moukalled, L. Mangani, *The finite volume method in computational fluid dynamics: An Advanced Introduction with OpenFOAM and Matlab*. New York: Springer, 2016.

[33] T. Poinsot and S. Lelef, "Boundary conditions for direct simulations of compressible viscous flows," *Journal of Computational Physics*, vol. 101, no. 1, pp. 104–129, 1992.

[34] D. G. Goodwin, *Cantera C++ user's guide*. California: California Institute of Technology, 2002.

[35] R. Kee, J. Grcar, M. Smooke, J. Miller, and E. Meeks, "Premix: A fortran program for modeling steady laminar one-dimensional premixed flames," *Sandia National Laboratories*, 1985.

Study questions

1. Why the flame front has a cone shape?
2. How will a new table with a different equivalence ratio affect the flame shape?
3. Why is the energy equation not needed for this model?
4. What kind of behavior would be expected from a model considering the energy equation?
5. What is the effect of flame geometry over flame speed?
6. How do the thermo-diffusivity instabilities affect the local flame speed?