FIAMMA USER GUIDE

Pietro Giraudi

Imperial College London, June 2023

1 Introduction

Welcome to the user guide for 'FIAMMA', a MATLAB solver specifically developed to calculate the Flame Transfer Function (FTF) for Jet Diffusion Flames. This is achieved through a direct integration of the governing equations across a range of frequencies.

The process implemented by the solver is methodical. It initiates by solving for the steady state solution, followed by the execution of forced simulations.

The estimation of the heat release rate involves the volumetric integration of the ideal gas internal energy. Subsequently, the gain and phase lag are computed by comparing this heat release rate signal with the velocity signal found at the domain inlet.

Operating within an axisymmetric domain, 'FIAMMA' adopts a rectangular mesh for the discretisation of the domain. The finite difference method is then employed to solve the governing equations within this domain.

'FIAMMA' offers a range of models to suit various needs. This includes models with constant density and diffusivity or variable diffusivity and density. This adaptability makes 'FIAMMA' versatile in its application to Flame Transfer Function estimation tasks.

'FIAMMA' was developed specifically for hydrogen-air flames but it accepts a variety inlet conditions such as different chemical compositions of the fuel and oxidiser stream, different domain dimensions and injection velocities.

The flame properties are defined in the flame structure, which is computed using the Cantera toolbox.

1.1 Problem geometry

The flame analysed in this script is an axisymmetric jet diffusion flame. As shown in Fig.1, the fuel is injected in the combustion chamber from the inner aperture, while the oxidiser stream enters the domain from the outer section. When the reactants come in contact, a flame is established starting from the interface between the fuel and oxidizer inlet.

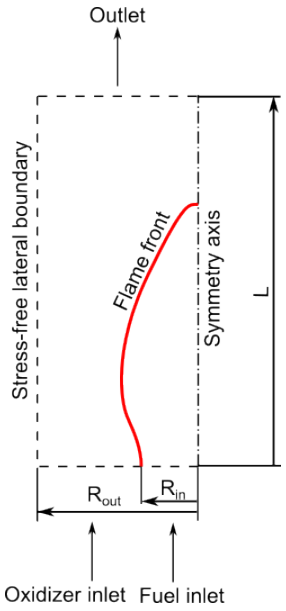

Figure 1: Sketch of the geometry.

2 Requirements

The solver requires the Cantera toolbox for MATLAB. Installation details can be found at the official Cantera website https://cantera.org/.

3 Usage

The solver main script is 'main.m'. From this script the user can define all the relevant inputs, and the solver functions are called.

3.1 Inputs

The solver inputs are identified by the '% $<$ $-$ $-$ $-$ $-$ $-$ $-$ $-$ $-$' comment.

The inputs are:

- R_in: radius of the fuel injector [m];
- R_out: domain radius [m];
- L: domain length [m];
- dx: spatial mesh discretisation in the radial direction [m];
- dz: spatial mesh discretisation in the axial direction [m];

- V_BC_F: injection velocity of the fuel stream [m/s];
- V_BC_O: injection velocity of the oxidiser stream [m/s];
- F_BC_F: mixture fraction value of the fuel stream (this value should be 1 by default);
- F_BC_O: mixture fraction value of the oxidiser stream (this value should be 0 by default);
- Amplitude: forcing amplitude (this value is normalised by the fuel injection velocity; a value of 0.01 means that the forcing amplitude is 1% of the fuel injection velocity);
- Frequency: forcing frequencies [Hz];
- force_Fuel: this boolean applies the forcing to the fuel stream;
- force_Oxi: this boolean applies the forcing to the oxidiser stream;
- Y_f_0: mass fraction of the fuel species in the fuel stream;
- Y_o_0: mass fraction of the oxidising species in the oxidiser stream;
- Y_p_0: mass fraction of the product species in the oxidiser stream;
- T_in: inlet temperature of the fuel and oxidiser stream [K];
- p: inlet pressure [Pa];
- tol_steady: tolerance parameter for the steady state solver;
- tol_forced: tolerance parameter for the unsteady solver;
- read_Solution: boolean to start the steady solver from a previously computed solution;
- read_Solution_Filename: string with the filename for the initial guess;
- plot_Grid: boolean to generate a plot with the mesh;
- plot_SteadyState: boolean to generate a plot with the steady solution;
- plot_Movie: boolean to plot the movie of the unsteady simulation;
- plot_FTF: boolena to generate a plot of the flame transfer function;
- disp_Stats: boolean to display the solver statistics;
- save_Solution: boolean to save the solution;

3.2 Functions

All the functions are stored in the 'FUNCTIONS' folder.

The functions are:

- BKFlameStructure(): function to generate a Burke-Schumann flame structure;
- CanteraH2(): function to generate a H2-Air flame structure with Cantera;
- CanteraCH4(): function to generate a CH4-Air flame structure with Cantera;
- computeTF(): function to compute the gain and phase lag;
- dx(): function to compute the first derivative in the radial direction;
- dz(): function to compute the first derivative in the axial direction;
- d2x(): function to compute the second derivative in the radial direction;
- d2z(): function to compute the second derivative in the axial direction;
- generateGrid(): function used to generate the MESH structure;
- getProblemData(): function to generate the PROBLEMDATA structure;
- my_Cp() / my_T() / my_D() / my_rho() / my_mu() / my_Y(): flame structure look up functions;
- plotFigure(): function to plot the flame shape;
- plotMovie(): function to plot the movie of the unsteady simulation;
- processFTF(): function to create, plot and save the flame transfer function;
- RK4(): fourth order Runge-Kutta integrating function (not used in the default solver);
- steadyBC(): function for generating the steady inlet boundary conditions;
- unsteadyBC(): function for generating the unsteady inlet boundary conditions;
- SteadyFun(): function that contains the governing equations for the SteadySolver() function;
- UnsteadyFun(): function that contains the governing equations for the UnsteadySolver() function;
- SteadySolver(): solver for the steady state solution;
- UnsteadySolver(): solver for the forced simulation.

3.3 Data Structures

3.3.1 MESH

MESH is the datastructure that contains the coordinates of the mesh points. It is generated by the generateGrid() function. It is saved in the 'MESH.mat' file. It contains:

- MESH.xx: radial coordinate;
- MESH.zz: axial coordinate;

3.3.2 FLAMESTRUCTURE

FLAMESTRUCTURE is the datastructure that contains the flame structure. It is generated by the functions BKFlameStructure()/CanteraH2()/CanteraCH4(). It is saved in the 'FLAMESTRUCTURE.mat' file. It contains:

- FLAMESTRUCTURE.Z: mixture fraction;
- FLAMESTRUCTURE.Z_st: stoichiometric mixture fraction;
- FLAMESTRUCTURE.T: temperature distribution;
- FLAMESTRUCTURE.YF: fuel mass fraction distribution;
- FLAMESTRUCTURE.YO: oxidiser mass fraction distribution;
- FLAMESTRUCTURE.YP: product mass fraction distribution;
- FLAMESTRUCTURE.YOH: OH radical mass fraction distribution;
- FLAMESTRUCTURE.YN: nitrogen mass fraction distribution;
- FLAMESTRUCTURE.Cp: specific heat distribution;
- FLAMESTRUCTURE.D: mass diffusivity coefficient distribution;
- FLAMESTRUCTURE.rho: density distribution;
- FLAMESTRUCTURE.mu: viscosity distribution;

3.3.3 PROBLEMDATA

PROBLEMDATA is the datstructure that contains the reference values used by the solver. It is generated by the getProblemData() function. It is saved in the 'PROBLEMDATA.mat' file.

3.4 Models available

To switch between different models the relevant lines must be uncommented/commented in both 'SteadyFun()' and 'UnsteadyFun()' functions.

3.4.1 Governing equations

- Mass conservation equation:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{u}) \tag{1}$$

in axisymmetric cylindrical coordinates:

$$\frac{\partial \rho}{\partial t} + \left(\frac{1}{r} \frac{\partial (r \rho U)}{\partial r} + \frac{\partial (\rho V)}{\partial z} \right) = 0 \tag{2}$$

- Species conservation equation:

$$\frac{\partial \rho Y_\alpha}{\partial t} = -\nabla \cdot (\rho Y_\alpha (\mathbf{u} + \mathbf{V}_\alpha)) + \dot{\omega}_\alpha \tag{3}$$

Assuming a reaction of the type

$$\nu_F(Fuel) + \nu_O(Oxidiser) \rightarrow \nu_P(Product) \tag{4}$$

it is possible to define s as the mass of $Oxidiser$ needed to burn a unit mass of $Fuel$

$$s = \frac{\nu_O W_O}{\nu_F W_F} \tag{5}$$

and therefore

$$\dot{\omega}_O = s \dot{\omega}_F \tag{6}$$

Define the normalised mixture fraction as

$$F = \frac{s Y_F - Y_O + Y_O^0}{s Y_F^0 + Y_O^0} \tag{7}$$

and substitute in the species conservation equation to obtain:

$$\frac{\partial \rho F}{\partial t} = -\nabla \cdot (\rho F (\mathbf{u} + \mathbf{V}_\alpha)) \tag{8}$$

with Fick's law and applying mass conservation equation:

$$\rho \left(\frac{\partial F}{\partial t} + \mathbf{u} \cdot \nabla F \right) = \nabla \cdot (\rho D \nabla F) \tag{9}$$

- Momentum conservation equation:

$$\frac{\partial (\rho \mathbf{u})}{\partial t} = -\nabla \cdot (\rho \mathbf{u} \mathbf{u}^\mathbf{T}) - \nabla p + \nabla \cdot \tau + \rho \mathbf{g} \tag{10}$$

neglect pressure, neglect gravity effects, neglect radial velocity and apply mass conservation

$$\rho \left(\frac{\partial U}{\partial t} + U \frac{\partial U}{\partial z} \right) = \mu \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial U}{\partial r} \right) + \frac{4}{3} \frac{\partial^2 U}{\partial z^2} \right] \tag{11}$$

3.4.2 Giraudi Model

-Mixture fraction equation:

$$\frac{\partial F}{\partial t} = -U^* \frac{\partial F}{\partial z^*} + \frac{1}{Pe}\left[D^* \frac{\partial^2 F}{\partial r^2} + \frac{1}{\rho^*}\frac{\partial(\rho^* D^*)}{\partial r^*}\frac{\partial F}{\partial r^*} + \frac{D^*}{r^*}\frac{\partial F}{\partial r^*} + D^* \frac{\partial^2 F}{\partial z^{*2}} + \frac{1}{\rho^*}\frac{\partial(\rho^* D^*)}{\partial z^*} + \frac{\partial F}{\partial z^*}\right] \tag{12}$$

-Velocity equations:

$$\frac{\partial U^*}{\partial t^*} = -U^* \frac{\partial U^*}{\partial z^*} + \frac{1}{Re}\frac{\mu^*}{\rho^*}\left[\frac{U^*}{\partial r^{*2}} + \frac{1}{r^*}\frac{\partial U^*}{\partial r^*} + \frac{4}{3}\frac{\partial^2 U^*}{\partial z^{*2}}\right] \tag{13}$$

where every variable has been normalised as $\phi = \phi_{ref} \cdot \phi^*$, and $Re = \frac{\rho_{ref} U_{ref} L_{ref}}{\mu_{ref}}$,$Pe = \frac{L_{ref} U_{ref}}{D_{ref}}$

3.5 Flame structure models available

The flame structures available are:

- Burke-Schumann flame structure: a Burke-Schumann flame structure can be generated with the 'BKFlameStructure()' function.
- Hydrogen counter flow diffusion flame: an hydrogen counter flow diffusion flame can be generated with the 'CanteraH2()' function.
- Methane counter flow diffusion flame: a methane counter flow diffusion flame can be generated with the 'CanteraCH4()' function.

To switch between different flame structures the relevant lines must be commented/uncommented from the 'main.m' script.

4 Example

The 'EXAMPLE' folders contains a solved case. You can find a fully-populated directory with all the output files and figures.

5 Contact information

For information about the code contact Pietro Giraudi at 'pietro.giraudi16@imperial.ac.uk'.