Proceedings of CFD with OpenSource Software, 2022
January 16th-20th, 2023, Chalmers University of Technology, Sweden

Implementation of FGM model for premixed flames in OpenFOAM

Rafael Becker Meier

Federal University of Santa Catarina, Brazil

Contents

Motivation and Objectives

Performance challenges

coupling fluid mechanics with chemical kinetics for combustion simulation [1]

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

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

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

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

Species equation issues

- N can have $\approx 10^1$ until $\approx 10^3$ species
- The diffusion terms become complex
- Required time step splitting
- $\dot{\omega}_i^{'''}$ can have different time scales, resulting in a stiffness problem for the ODE solver

[1] Oijen *et al*. Progress in Energy and Combustion Science, 2016.

Motivation and Objectives

Roads have taken to skip from complex chemistry

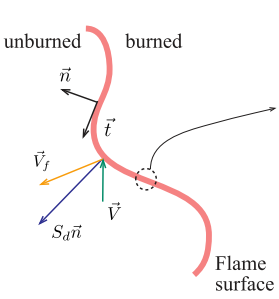
Chemical reduction techniques:

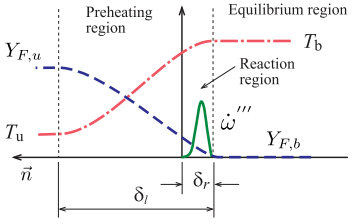
- Conventional chemical mechanisms reduction (DRG, DSA, virtual chemistry, etc.)
- ILDM: Intrinsic Low-Dimensional Manifolds
- CSP: Computational Singular Perturbation
- FGM: Flamelet-Generated Manifold

Objective:

Implementation of the FGM model in OpenFOAM to solve premixed combustion problems for laminar and direct numerical simulations.

Theory: *Premixed flame structure and propagation speed*





Flame structure

Flame speeds

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

Flat flame speed

$$S_d = (\vec{V}_f - \vec{V}) \cdot \vec{n} = S_{reac} + S_{diff}$$

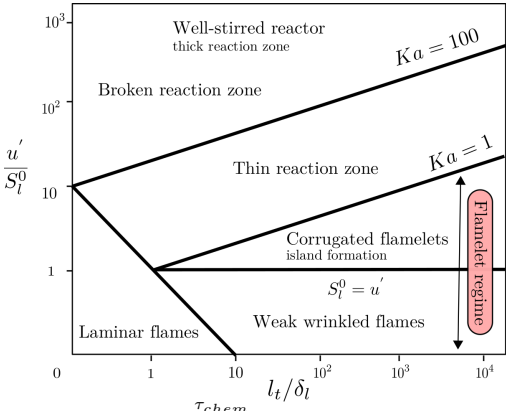
$$S_l^0 = \tilde{S}_d = \frac{\rho}{\rho_u} S_d$$

Stretched flame speed

$$S_d = (S_{reac} + S_{diff})\vec{n} - S_{\vec{t}}$$

$$\tilde{S}_d = \frac{\rho}{\rho_u}\left((S_{reac} + S_{diff})_{\vec{n}} - S_{\vec{t}}\right)$$

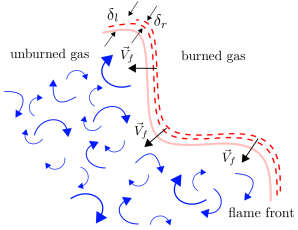
Theory: *Combustion regimes - Borghi-Peters diagram*



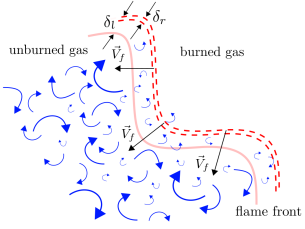
$$\mathrm{Ka} = \frac{\tau_{chem}}{\tau_\eta}$$

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

Weak wrinked flames



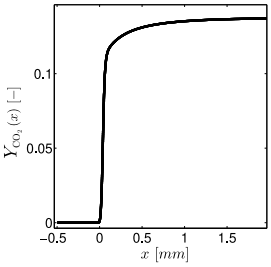
Corrugate flamelets



FGM model: *Big picture*

- 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.
- The set of combustion thermo-chemistry variable, e.g., `temperature, density and reaction rate`, are parameterized as a function of specific variables controlling the temporal evolution of the combustion process.

Premixed flame structure of a methane/air mixture computed with CHEM1D solver and GRI3.0 reaction mechanism $T_u = 298$ K $p = 1$ bar [2].



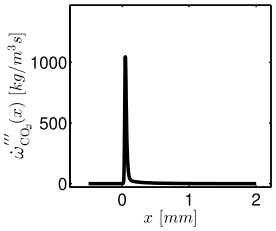
The scaled progress variable c is defined as follows [2]

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

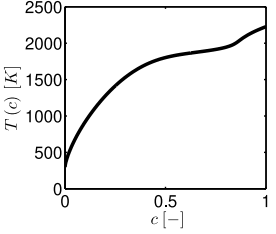
[2] Souza *et al*. Journal of Turbulence, 2017.

Manifolds construction [2]

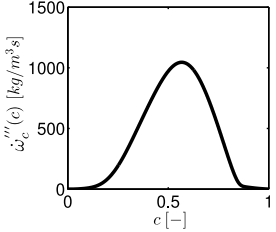
Source term of CO2 as function of x-axis in physical space.



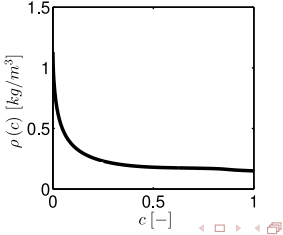
Temperature as a function of c.



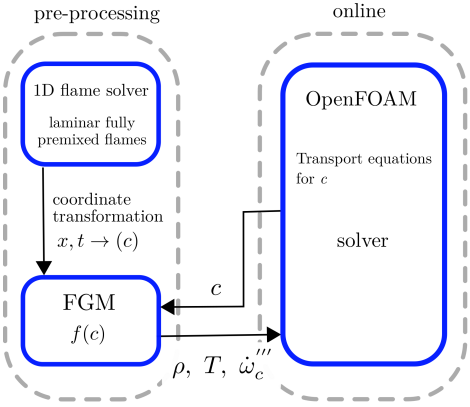
Source term of CO2 as function of c.



Density of the mixture as a function of c.



Interface procedure between the FGM-CFD



The transport equation and the thermo-chemistry states are defined as

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

$$\rho = f_1(c)$$

$$T = f_2(c)$$

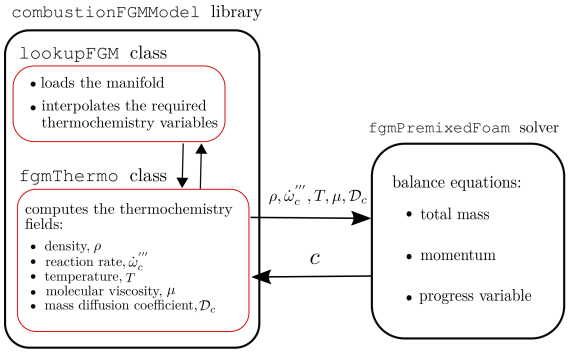
$$\dot{\omega}_c''' = f_3(c)$$

FGM-PremixedFoam solver: *code workflow*

The FGM solver, `fgmPremixedFoam`, is built from the `rhoReactingBuoyantFoam` where a new library named `combustionFGMModel` is called by the OpenFOAM.

New objects created from new classes:

- `lookupFGM fgmTable(mesh);`
- `fgmThermo thermo(mesh, fgmTable, p, PV);`



FGM-PremixedFoam solver

Top-level changes in the `rhoReactingBuoyantFoam`

In `rhoReactingBuoyantFoam` solver, the thermo object is declared in createFields.H. In this solver, the lines below are replaced by the following code observed on the left. The progress variable, c, is declared as PV in line 4, and the pressure field, p, is declared in line 17.

`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();
```

`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 );
```

FGM-PremixedFoam solver

Top-level changes in the `rhoReactingBuoyantFoam`

The objects related to classes `lookupFGM` and `fgmThermo` are initialized in lines 32 and 35, respectively.

`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();
```

`fgmPremixedFoam/createFields.H`

```
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);
```

FGM-PremixedFoam solver

Top-level changes in the `rhoReactingBuoyantFoam`

The FGM model simplifies the species and the energy equation. Hence, lines 107 and 108 which are default in `rhoReactingBuoyantFoam.C`, code in left column, are replaced by the solution of `PVEqn.H` in `fgmPremixedFoam.C`, column in the left.

`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             {
```

`fgmPremixedFoam/fgmPremixedFoam.C`

```
89  #include "rhoEqn.H"
90
91          while (pimple.loop())
92          {
93              #include "UEqn.H"
94              #include "PVEqn.H"
95
96              // --- Pressure corrector loop
97              while (pimple.correct())
98              {
```

FGM-PremixedFoam solver

Top-level changes in the `rhoReactingBuoyantFoam`

fgmPremixedFoam/UEqn.H

```
3     fvVectorMatrix UEqn
4     (
5         fvm::ddt(rho, U) + fvm::div(phi, U) + MRF.DDt(rho, U)
6       - fvm::laplacian(thermo.mu(), U)
7       - fvc::div(thermo.mu()*Foam::dev2(Foam::T(fvc::grad(U))))
8       ==
9       fvOptions(rho, U)
10    );
```

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

Within the scope of DNS for turbulence modeling is recommended:

- second-order `backward` difference method for time.
- third-order schemes for gradients, divergence and laplacian terms, e.g., `Gauss cubic`.
- third-order spatial interpolation scheme for faces flux and pressure, , e.g., `cubic`.
- To prevent numerical instabilities arising from the discretization schemes, the time step should dynamically adjusted to retain the convective $\texttt{CFL} \leq 0.1$.

FGM-PremixedFoam solver

Top-level changes in the `rhoReactingBuoyantFoam`

The progress variable `PV` is solved as a transport equation as seen in the code aside. 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()`.

`fgmPremixedFoam/PVEqn.H`

```
12 tmp<fvScalarMatrix> tPVEqn
13     (
14     (
15       fvm::ddt(rho, PV)
16     + fvm::div(phi, PV)
17       - fvm::laplacian(thermo.Dmass(), PV)
18       == thermo.sourcePV()
19         )
20     );
21
22     fvScalarMatrix& PVEqn = tPVEqn.ref();
23
24     PVEqn.relax();
25     fvOptions.constrain(PVEqn);
26
27     PVEqn.solve();
28     fvOptions.correct(PV);
29     PV = max(min( PV, 1.0 ), 0.0 );
30
31     T = thermo.T();
32
33     thermo.correct();
```

Thermophysical library: *lookupFGM*

`combustionFGMModel/lookupFGM.C`

```
32 lookupFGM::lookupFGM
33 (
34  const fvMesh& mesh
35 )
36   :
37   IOdictionary
38   (
39     IOobject
40     (
41       "fgmProperties",
42       mesh.time().constant(),
43       mesh,
44       IOobject::MUST_READ_IF_MODIFIED,
45       IOobject::NO_WRITE
46       )
47   ),
48   mesh_(mesh),
49   PV_table(lookup("PV")),
50   sourcePV_table(lookup("sourcePV") ),
51   T_table(lookup("T") ),
52   rho_table(lookup("rho"))
53
54 ...
```

Simulation case directory

```
.
|---0
|  |-- inletPoiseulleFlow
|  |-- p
|  |-- p_rgh
|  |-- PV
|  |-- U
|---Allrun
|---constant
|  |-- fgmProperties
|  |-- g
|  |-- thermophysicalProperties
|---system
   |-- blockMeshDict
   |-- controlDict
   |-- decomposeParDict
   |-- fvSchemes
   |-- fvSolution
```

Thermophysical library: *lookupFGM* manifold

Manifold for stochiometric mixture of CH_4/air at atmospheric conditions

constant/fgmProperties

```
32 FoamFile
33 {
34     version     2.0;
35     format      ascii;
36     class       dictionary;
37     location    "constant";
38     object      fgmProperties;
39 }
40 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
41
42 PV
43 (
44 0
45 0.00125
46 0.0025
47 0.00375
48
49 ...
50
51 0.99625
52 0.9975
53 0.99875
54 1
55 );
```

constant/fgmProperties

```
51 0.99625
52 0.9975
53 0.99875
54 1
55 );
56
57 T
58 (
59 298
60 309.793419010089
61 316.934133757152
62 323.161361068549
63
64 ...
65
66 2227.11248859335
67 2226.90945781197
68 2226.0606337391
69 2224.10100091593
70 );
```

Thermophysical library: *lookupFGM*

Linear interpolation member function

The `lookupFGM` member function, `lookupFGM::interpolateValue1D`, computes the interpolation process of ρ, T and the source term, $\dot{\omega}_c^{'''}$, as the progress variable c/`PV` is solved in `PVEqn.H`. The linear inteporlation is performed according to Eq. below

$$y_i = (c_i - c_{j-1}) \cdot \frac{y_j - y_{j-1}}{c_j - c_{j-1}} + y_{j-1},$$

where i and j stand for the values in the cells, and the values in the manifold, respectively.

`lookupFGM/lookupFGM.C`

```
32 Foam::scalar Foam::lookupFGM::interpolateValue1D
33 (
34     const List<scalar>& table,
35     scalar pvValue,
36     const List<scalar>& pvTable
37 ) const
38 {
39
40     ...
41
42         // INTERPOLATION ALGORITHM
43         for(int j=0; j < pvTable.size(); j++ )
44          {
45            ...
46          }
47
48      rate = (upper_table - lower_table )/ \
49          max((upper_pvTable - lower_pvTable),smallValue);
50
51          interpolatedValue = ( pvValue - lower_pvTable )*rate +
            lower_table;
52
53          interpolatedValue = max(interpolatedValue,min(table));
54
55         }
56
57     return interpolatedValue;
58
59 }
```

Thermophysical library: *fgmThermo*

List of member data in the `fgmThermo` class

```
//- FGM Model
word fgmThermoModel_;

//- Lewis number [-]
const scalar Le_;

//- Constant of gas mixture [J/Kg K]
const scalar Rgas_;

//- Look up table and routines
const lookupFGM& fgmTable_;

//- Pressure [Pa]
volScalarField p_;

//- Progress variable [-]
volScalarField& PV_;

//- Temperature [K]
volScalarField T_;

//- Density [Kg/m3]
volScalarField rho_;

//- Compressibility at constant temperature [s2/m2]
volScalarField psi_;

//- Mass diffusivity [Kg/m s]
volScalarField Dmass_;

//- Viscosity [Kg/m s]
volScalarField mu_;

//- Rate reaction of progress variable [Kg/m3s]
volScalarField sourcePV_;

//- Should the dpdt term be included in the enthalpy
    ↪ equation
Switch dpdt_;
```

Thermophysical library: *fgmThermo*

combustionFGMModel/fgmThermo/fgmThermo.C

```
32 Foam::fgmThermo::fgmThermo
33 (
34     const fvMesh& mesh,
35     const lookupFGM& lookupFGM,
36     volScalarField& p,
37     volScalarField& PV
38 )
39 :
40   IOdictionary
41   (
42     IOobject
43     (
44      "thermophysicalProperties",
45      mesh.time().constant(),
46      mesh,
47      IOobject::MUST_READ_IF_MODIFIED,
48      IOobject::NO_WRITE
49      )
50   ),
51   fgmThermoModel_(lookup("thermoType")),
52   Le_(lookupOrDefault<scalar>("Le",1.0)),
53   Rgas_(lookupOrDefault<scalar>("Rgas",287.0)),
54
55   fgmTable_(lookupFGM),
56   p_(p),
57   PV_(PV),
58
59    ...
60
61   correct();
```

constant/thermophysicalProperties

```
32 FoamFile
33 {
34     version     2.0;
35     format      ascii;
36     class       dictionary;
37     location    "constant";
38     object      thermophysicalProperties;
39 }
40 // * * * * * * * * * * * * * * * * * * * * //
41
42 thermoType fgm1DModelDNS;
43
44 // Lewis number of CO2
45 Le 1.384;
46
47 // Ideal gas mixture constant [J/Kg K]
48 Rgas 287.05;
49
50 // Pressure-work
51 dpdt false;
52
53 // ********************************* //
```

Thermophysical library: *fgmThermo*

`fgmThermo::correct()`

The `correct()` member function is responsible for updating all variables as the transport equation for `PV` is solved.

The members data, `T_`, `rho_`, and `sourcePV_` are corrected for each cell using the interpolation method.

The transport properties, `Dmass_`, `mu_` and `psi_`, are calculated according to specific the member functions.

The members data are also updated for the patches.

`combustionFGMModel/fgmThermo/fgmThermo.C`

```
32  void Foam::fgmThermo::correct()
33  {
34      scalarField& TCells = T_.primitiveFieldRef();
35      scalarField& psiCells = psi_.primitiveFieldRef();
36      scalarField& sourcePVCells = sourcePV_.primitiveFieldRef();
37
38      ...
39
40      // Interpolate for internal field
41      forAll(TCells, celli)
42      {
43          TCells[celli] = fgmTable_.interpolateValue1D
44                          (
45                              fgmTable_.T_table,
46                              PVCells[celli],
47                              fgmTable_.PV_table
48                          );
49
50          DmassCells[celli] = massDiffusivity_model(TCells[celli]);
51
52          muCells[celli] = viscosity_model(TCells[celli]);
53
54          ...
55
56          psiCells[celli] = compressibility_model(rhoCells[celli], pCells[celli]);
57
58          ...
```

Thermophysical library: *fgmThermo*

Transport properties

Compressibility at constant temperature [s2/m2], `psi_`:

$$\psi = \frac{\rho}{p}$$

Mass diffusivity [Kg/m s], `Dmass_`:

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

$$\mathcal{D}_c = \frac{\lambda}{c_p \mathrm{Le}_{\mathrm{CO}_2}}$$

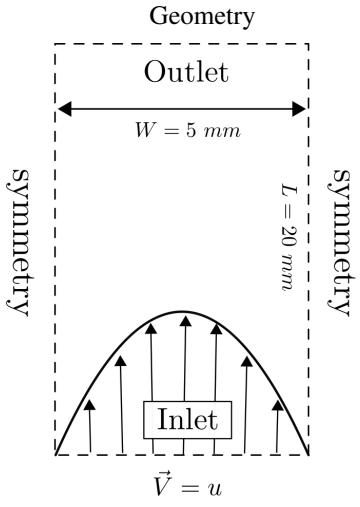
Viscosity [Kg/m s], `mu_`:

$$\mu = \mu_{ref} \left(\frac{T}{T_{ref}}\right)^{1.5} \frac{T_{ref} + S}{T + S}$$

combustionFGMModel/fgmThermo/fgmThermo.C

```
32 // * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
33
34 Foam::scalar Foam::fgmThermo::compressibility_model(const scalar rho, const scalar p) const
35 {
36   return rho/p;
37 }
38
39 Foam::scalar Foam::fgmThermo::massDiffusivity_model(const scalar T) const
40 {
41
42   scalar T298 = 298;
43   scalar C069 = 0.69;
44   scalar CD = 2.58E-5;
45
46   return CD*pow(T/T298,C069)/Le_;
47
48 }
49
50 Foam::scalar Foam::fgmThermo::viscosity_model(const scalar T) const
51 {
52   // Model: Sutherland's law
53   scalar muRef = 1.7894E-5;
54   scalar TRef = 273.15;
55   scalar S = 110.4;
56
57   return muRef*pow(T/TRef,1.5)*((TRef+S)/(T+S));
58
59 }
```

Test case: *2D Bunsen flame*



Geometry, initial and boundary conditions:

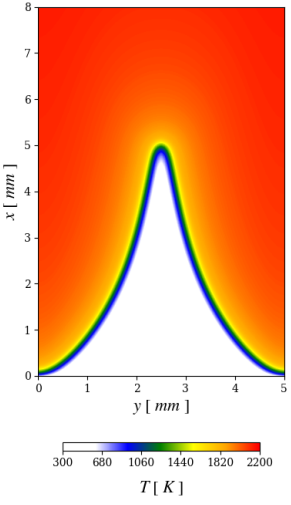
- Domain dimensions: $5x20$ mm
- Mesh size: $60{\cdot}10^3$ cells
- $c(t = 0) = 0.8$ in the entire domain
- Boundary conditions
  1. Prescribed Poiseuille inlet velocity
  2. $c = 0$ at the inlet
  3. Symmetric BCs at the sides of the domain
  4. Partially non-reflective BCs (PNRBC)

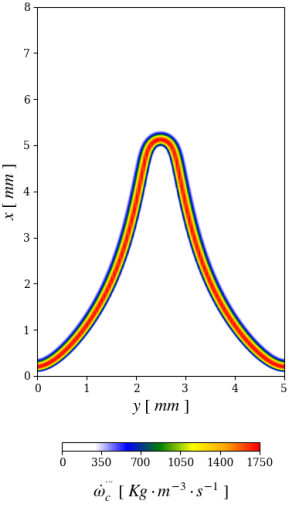
Test case: *2D Bunsen flame*

Video of the flame front time evolution (it works on Okular)

Test case: *2D Bunsen flame*

Steady-state solution





Results and discussions: *Performance comparison*

Comparison with EBI-DNS
(Engler-Bunte-Institute, KIT, Germany)
EBI-DNS solves the fully compressible Navier-Stokes, species and energy equations for reacting gas mixtures coupled to the chemical kinetics library Cantera [3].

The reduced mechanism of Kee *et al.*[4] with 17 species and 58 elementary reactions was used to run the EBI-DNS case.

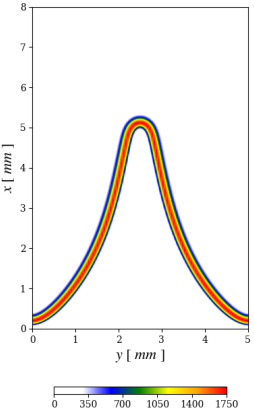
Results:
The solution is not equal due to the simplifications of the model.

EBI-DNS running in a HPC structure with 20 cores in parallel, takes 2 days to reach the steady-state solution.

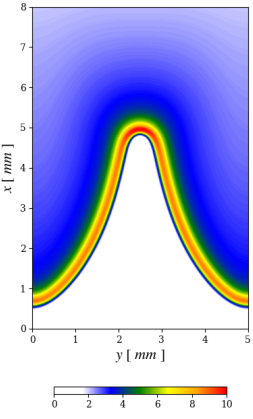
FGM-PremixedFoam running in a conventional laptop with 4 cores in parallel, takes around 1 hour to reach the steady-state solution.

[3] Zirwes *et al.*, Flow, Turbulence and Combustion, 2020.
[4] Kee *et al.*, Sandia National Laboratories, 1985.

FGM-PremixedFoam
Mesh size: $60{\cdot}10^3$ cells



EBI-DNS
Mesh size: $450{\cdot}10^3$ cells



THANK YOU FOR YOUR ATTENTION