Applied Mechanics Department
Chalmers University of Technology

CFD with OpenSource software, assignment 3

Tutorial XiFoam

Version: OpenFOAM-1.7.x

Author:
Ehsan Yasari

Peer reviewed by:
Josef Runsten
Jelena Andric

November 4, 2010

Contents

List of Figures

List of Tables

Chapter 1

Tutorial XiFoam

1.1 Introduction

In order to help the users to understand the physics of the premixed turbulent combustion, a short introduction of this phenomenon is presented. Besides, this tutorial looks into the implementation of the code, and presents a brief explanation of different parameters as well as different classes used in this solver. Moreover, it describes how to pre-process and run a premixed turbulent combustion case using the `XiFoam` solver. Finally, an implementation of new combustion model is presented. The case study is a cubic combustion chamber, with ignition at its centre, at 0.001 ms.

1.2 Physics of Premixed Combustion

This section presents a brief introduction to the modelling of the premixed turbulent combustion and its governing equations.
In premixed combustion, fuel and oxidizer are mixed at the molecular level prior to ignition. Combustion occurs as a flame front propagates into the unburnt reactants. Examples of premixed combustion include aspirated internal combustion engines, lean-premixed gas turbine combustors, and gas-leak explosions.
The effect of turbulence is that it wrinkles and stretches the propagating laminar flame sheet, increasing the sheet area and, in turn, the effective flame speed. The large turbulent eddies tend to wrinkle and corrugate the flame sheet, while the small turbulent eddies, if they are smaller than the laminar flame thickness, may penetrate the flame sheet and modify the laminar flame structure.
As the premixed flame is a reaction wave propagating from burned to fresh gases, the basic parameter is known to be the progress variable. In the fresh gas, the progress variable is conventionally put to zero. In the burned gas, it equals unity. Across the flame, the intermediate values describe the progress of the reaction. A progress variable can be set with the help of any quantity, like temperature, reactant mass fraction, provided it is bounded by a single value in the burned gas and another one in the fresh gas. The progress variable is usually named c, in usual notations[1].

$$c = \frac{T - T_f}{T_b - T_f} \tag{1.1}$$

Where `b` stands for burned gas, and `f` stands for fressh gas. It is seen that c is a normalization of a scalar quantity.
In OpenFOAM, the flame front propagation is modelled by solving a transport equation for the density-weighted mean reaction regress variable denoted by b(eq 1.7), where:

$$b = 1 - c \tag{1.2}$$

$$\frac{\partial}{\partial t}(\rho b) + \nabla.(\rho \vec{u} b) - \nabla.(\frac{\mu_t}{Sc_t}\nabla b) = -\rho S_c \tag{1.3}$$

$Sc_t = \frac{\mu}{\rho D}$: turbulent Schmidt number.
S_c : reaction regress source term (the dimesnion is $[T^{-1}]$), and is modeled as equation 1.4:

$$\rho S_c = \rho_u S_u \Xi |\nabla b| \tag{1.4}$$

By substituting equation 1.4 to equation 1.7 we would have:

$$\frac{\partial}{\partial t}(\rho b) + \nabla.(\rho \vec{u} b) - \nabla.(\frac{\mu_t}{Sc_t}\nabla b) = -\rho_u S_u \Xi |\nabla b| \tag{1.5}$$

where:

- b : mean reaction regress variable
- S_u : laminar flame speed $[m/s]$
- D : diffusion coefficient $[m^2/s]$
- Ξ : Turbulent flame velocity and laminar flame velocity ratio
- ρ_u : density of unburnt mixture $[kg/m^3]$

Based on this definition:

b=1 : unburnt mixture
b=0 : burnt mixture

The value of b is defined as a boundary condition at all flow inlets. It is usually specified as either 0 (unburnt) or 1 (burnt). OpenFOAM has a premixed turbulent combustion model based on the reaction regress variable(b=1-c) approach. Information about this model is provided in the `bEqn.H` file of the `XiFoam` solver.

1.3 Introduction of the XiFoam files

In this section, the XiFoam solver and some of the important related files of the solver will be explained. XiFoam, is a solver for compressible premixed/partially-premixed combustion with turbulence modelling. At first we should create our own XiFoam solver and rename it to myXiFoam.

1.3.1 Creating myXiFoam solver

Follow the instruction:

```
cd $WM_PROJECT_USER_DIR
cp -r $FOAM_APP/solvers/combustion/XiFoam myXiFoam
cd myXiFoam
```

We should rename `XiFoam.C` to `myXiFoam.C` and also `bEqn.H` to `myBEqn.H`:

```
mv XiFoam.C myXiFoam.C
mv bEqn.H myBEqn.H
```

Now, we also have to modify the `files` in `Make` directory:

```
sed -i s/"XiFoam"/"myXiFoam"/g Make/files
sed -i s/"FOAM_APPBIN"/"FOAM_USER_APPBIN"/g Make/files
```

So we would have:

```
myXiFoam.C
EXE = $(FOAM_USER_APPBIN)/myXiFoam
```

Also we have to replace `bEqn.H` with `myBEqn.H` in all files:

```
sed -i s/"bEqn.H"/"myBEqn.H"/g *.*
```

Now run wmake command and you will have `myXiFoam` solver:

```
wmake
```

1.3.2 Add subroutine to calculate the flame propagating radius(Optional)

In premixed turbulent combustion, flame propagation radius is one of the important parameters which help to better understanding the flame treatment. So it should be measured during the simulation. The expression for the radius is written in equation 1.6:

$$R = [(\frac{3}{4\pi\rho_b}) \iiint \rho(1-b)dxdydz]^{1/3} \tag{1.6}$$

To calculate this parameter we must do the following steps:
Create the `radiusFlame.H` file

```
gedit radiusFlame.H
```

In order to implement the equation 1.6, the integrate must be discretized. This done by summing the value of $\rho * (1-b)$ multiply the mesh volume($dxdydz$) in the whole domain.
Add the following lines in this file:

```
Info<< "Reading radiusFlame.H file "<<endl;
#include "mathematicalConstants.H"
volVectorField centres = mesh.C();
scalar SummationRho=0.0;
```

```
scalar RadiusMinRho=0.0;
const scalar coeff=3./(4.*mathematicalConstant::pi);
forAll(centres,k) { SummationRho=SummationRho+
                    (mesh.V()[k]*rho[k]*(scalar(1.)-b[k]))/(min(rho).value());
                  }
RadiusMinRho=Foam::pow(coeff*SummationRho,(1./3.));
Info<< "RadiusMinRho = "<< RadiusMinRho  <<endl;
```

Since we are going to calculate the radius at each time step, we include this file in `myXiFoam.C` time loop after the line: `runTime.write();`
So we will have:

```
runTime.write();
#include "radiusFlame.H"
```

Since we are going to save these data during the simulation, we should create a file and save them there. Do the following steps:

```
gedit createXiFoamOutput.H
```

Add the following line in this file:

```
OFstream RadiusFlame("XiFoamOutput.txt");
```

As `OFstream` command used, it is necessary to include some header files. That is done by adding the:

```
#include "IFstream.H"
#include "OFstream.H"
```

after:

```
#include "Switch.H"
```

in `myXiFoam.C` file.

Moreover, it is necessary to create `writeXiFoamOutput.H` file, and write:

```
RadiusFlame << "Time= "<< runTime.timeName() << "\tRadiusMinRho= "<< RadiusMinRho<<"
\tMin(rho)= "<< min(rho).value() <<endl;
```

Now :

```
#include "createXiFoamOutput.H"
```

Must be included in `myXiFoam.C` in the `main` function, and befor the `while` loop.
It is necessary to save the radius in the file after the calculation. That is done by adding the:

```
#include "writeXiFoamOutput.H
```

After the line:

```
#include "radiusFlame.H"
```

Finally run the wmake command:

```
wmake
```

Therefore, the following files are in `myXiFoam` directory:

```
|-- Make
| |-- files
| |-- linux64GccDPOpt
| |-- options
|-- UEqn.H
|-- createFields.H
|-- createXiFoamOutput.H
|-- ftEqn.H
|-- hEqn.H
|-- huEqn.H
|-- myBEqn.H
|-- myXiFoam.C
|-- myXiFoam.dep
|-- pEqn.H
|-- radiusFlame.H
|-- readCombustionProperties.H
‘-- writeXiFoamOutput.H
```

In the following section we look through some of these file.

1.3.3 myBEqn.H File

Following items are in `myBEqn.H` file:

- Transport equation for regress variable b
- Laminar flame speed based on the different models
- Weller combustion modell[2] for calculation Xi=St/Su

Transport equation for b is presented in equation 1.7 [2]:

$$\frac{\partial}{\partial t}(\rho b) + \nabla.(\rho \vec{u} b) - \nabla.(\frac{\mu_t}{Sc_t}\nabla b) = -\rho_u S_u \Xi |\nabla b| \tag{1.7}$$

and here is the implementation of this equation:

```
fvScalarMatrix bEqn
(
fvm::ddt(rho, b)
+ mvConvection->fvmDiv(phi, b)
+ fvm::div(phiSt, b, "div(phiSt,b)")
- fvm::Sp(fvc::div(phiSt), b)
- fvm::laplacian(turbulence->alphaEff(), b)
);
```

There are three models to calculate the `Xi`:

```
1- fixed
2- algebraic
3- transport
```

In the following parts these models will be discussed.

1- fixed:
Do nothing, Xi is fixed!

2- algebraic:
Algebraic equation is implemented based on the equation 1.8 and 1.9:

$$\Xi^*_{eq} = 1 + 0.62\sqrt{\frac{u'}{S_u}}R_\eta \tag{1.8}$$

$$\Xi_{eq} = 1 + 2(1-b)(\Xi^*_{eq} - 1) \tag{1.9}$$

where:

u' is the turbulence intensity.
R_η is the kolmogorov Reynolds number.

And the implementation is:

```
Xi == scalar(1) +
(scalar(1) + (2*XiShapeCoef)*(scalar(0.5) - b))
*XiCoef*sqrt(up/(Su + SuMin))*Reta;
```

`XiShapeCoef` and `XiCoef` are input data which discussed later in section 1.4.8

3- transport:
For mathematical formulation please refer to [2], and for implementation of this model you can refer to `bEqn.H` file.

1.4 Setting up the Case

This section provides necessary setup to run a case using the XiFoam(or myXiFoam) solver. Now we can run the case using myXiFoam solver which is exactly the same as XiFoam solver, except that we have just implemented additional piece of code to capture the flame radius.
Figure 1.1 presents a case study which is a cubic combustion chamber, and ignition occurred at its centre. Since the ignition is spherical, and the problem is symmetrical, to reduce the computational cost, 1/8 domain will be solved using symmetric boundary condition.

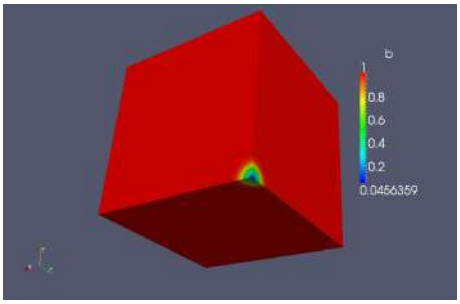


Figure 1.1: Geometry of the XiFoam tutorial case

1.4.1 Create the Geometry

You can download the case file from:
`http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2010/ehsanYasari/ehsanYasariFiles.tgz`

Alternatively, you can copy the default case of XiFoam to `run` directory and modify it.

```
run
cp -r $FOAM_TUTORIALS/combustion/XiFoam/ras/moriyoshiHomogeneous chamber $
cd chamber
```

As you can see, the file structure for XiFoam solver is similar to the other OpenFOAM tutorials. The case directory consists of the following subdirectories:

```
/0
/constant
/system
```

Run the following command to see an overview of the files in this case.

```
tree -L 2
```

In the following section we will look through these folders to initiate our setup.

1.4.2 Constant Folder

In the constant folder, the following files and directories are found:

```
|-- RASProperties
|-- combustionProperties
|-- g
|-- polyMesh (Folder)
|-- thermophysicalProperties
‘-- turbulenceProperties
```

1.4.3 polyMesh

The case domain consists of a square with length equals to 35mm. A uniform mesh of 35*35*35 is used to have cell size equals to 1mm. Since the cell size is smaller than the ignition diameter (3mm) we can capture the ignition.
We must modify the `blockMehDict` as follows:

```
gedit constant/polyMesh/blockMeshDict
```

Copy the new geometry in this file. Here are the `blockMesh` entries for this case:

```
convertToMeters 0.001;
vertices
(
    (0 0 0)                  //vertex No.1
    (0 35 0)                 //vertex No.2
    (35 0 0)                 //vertex No.3
    (35 35 0)                //vertex No.4
    (0 0 35)                 //vertex No.5
    (0 35 35)                //vertex No.6
    (35 0 35)                //vertex No.7
    (35 35 35)               //vertex No.8
);
blocks
(
    hex (0 2 3 1 4 6 7 5) (35 35 35) simpleGrading (1 1 1) //Block No.1
);
edges
(
);
patches
(
    symmetryPlane left
    (
        (0 4 5 1)
    )
    symmetryPlane right
    (
        (2 3 7 6)
    )
    symmetryPlane top
    (
        (1 5 7 3)
    )
    symmetryPlane bottom
    (
        (0 2 6 4)
    )
    symmetryPlane front
    (
        (4 5 7 6)
    )
    symmetryPlane back
    (
        (0 1 3 2)
    )
);
```

```
mergePatchPairs
 (
);
```

Then run the `blockMesh` command to generate your geometry and mesh:

```
blockMesh
```

To view the geometry in paraview, type:

```
paraFoam
```

We set other parameters in the constant directory in the next sections.

1.4.4 turbulenceProperties

In the turbulenceProperties file you can specify your turbulence model. You have two options :

1- `RASModel`
2- `LESModel`

In this tutorial we use `RASModel`, so the simulation type is set to `RASModel`:

```
simulationType RASModel;
```

With `RASModel` selected in this case, the choice of RAS model is specified in a `RASProperties` file.

1.4.5 RASProperties

In `RASProperties` file you can define your RAS model for compressible flows. You can choose the model from the table1.1.

keyword	description
laminar	Dummy turbulence model for laminar flow
kEpsilon	Standard k-ϵ model
kOmegaSST	k-ω-SSt model
RNGkEpsilon	RNG k-ϵ model
LaunderSharmaKE	Launder-Sharma low-Re k-ϵ model
LRR	Launder-Reece-Rodi RSTM
LaunderGibsonRSTM	Launder-Gibson RSTM
realizableKE	Realizable k-ϵ model
SpalartAllmaras	Spalart-Allmaras 1-eqn mixing-length model

Table 1.1: RAS turbulence models for compressible fluids-compressibleRASModels

For more information regarding RAS models one can refer to [3]. Here the `LaunderSharmaKE` model is selected:

```
RASModel LaunderSharmaKE;
turbulence on;
printCoeffs on;
```

1.4.6 g

In the `g` file you can specify the gravity.

```
dimensions [0 1 -2 0 0 0 0];
value ( 0 0 0 );
```

The gravity is set to zero in all directions.

1.4.7 thermophysicalProperties

The `thermophysicalProperties` file for the present case is:

```
thermoType hhuMixtureThermo<homogeneousMixture<sutherlandTransport<specieThermo<
    janafThermo<perfectGas>>>>>;
stoichiometricAirFuelMassRatio stoichiometricAirFuelMassRatio [ 0 0 0 0 0 0 0 ] 15.675;
fuel fuel 1 44.0962 200 5000 1000 7.53414 0.0188722 -6.27185e-06 9.14756e-10
-4.78381e-14 -16467.5 -17.8923 0.933554 0.0264246 6.10597e-06 -2.19775e-08
9.51493e-12 -13958.5 19.2017 1.67212e-06 170.672;
```

Note: this file also have the similar coefficient for `oxidant`, `reactant`, `product` and `burntProduct`.

⋆ keyword: `thermoType`:
The `thermophysicalProperties` dictionary is read by any solver that uses the thermophysical model library. A thermophysical model is constructed in OpenFOAM as a pressure-temperature (p - T) system from which other properties are computed. There is one compulsory dictionary entry called `thermoType` which specifies the complete thermophysical model that is used in the simulation.

The thermophysical modelling starts with a layer that defines the basic equation of state (here: perfectGas) and then adds more layers of modelling that derive properties from the previous layer(s). The naming of the `thermoType` reflects these multiple layers of modelling as listed in figure 1.2. Each layer have various options, so for more details you can refer to userGuide[3]. In the figure 1.3

	Type	Description
Thermophysical model	*hhuMixtureThermo*	Calculates enthalpy for unburnt gas and combustion mixture
Mixture properties	*egrMixture*	Exhaust gas recirculation mixture
	homogeneousMixture	Combustion mixture based on normalised fuel mass fraction b
	inhomogeneousMixture	Combustion mixture based on b and total fuel mass fraction f_t
	veryInhomogeneousMixture	Combustion mixture based on b , f_t and unburnt fuel mass fraction f_u
Transport properties	*constTransport*	Constant transport properties
	sutherlandTransport	Sutherland's formula for temperature-dependent transport properties
Derived thermophysical properties	*specieThermo*	Thermophysical properties of species, derived from C_p h, and/or s
Basic thermophysical properties	*hConstThermo*	Constant specific heat C_p model with evaluation of enthalpy h and entropy s
	janafThermo	C_p evaluated by a function with coefficients from JANAF thermodynamic tables, from which h , s are evaluated
Equation of State	*perfectGas*	Perfect gas equation of state

Figure 1.2: Description of the thermo entry

the possible entry for `XiFoam` solver is presented.

Figures 1.4 is the Inheritance diagram for `hhuMixtureThermo`. The diagram shows that `hhuCombustionThermo`

Possible entry for thermoType in thermophysicalProperties for XiFoam					
Thermophysical model	**Mixture properties**	**Transport properties**	**Derived thermophysical properties**	**Basic thermophysical properties**	**Equation of State**
hhuMixtureThermo	egrMixture	constTransport	specieThermo	hConstThermo	perfectGas
hhuMixtureThermo	egrMixture	sutherlandTransport	specieThermo	janafThermo	perfectGas
hhuMixtureThermo	homogeneousMixture	constTransport	specieThermo	hConstThermo	perfectGas
hhuMixtureThermo	homogeneousMixture	sutherlandTransport	specieThermo	janafThermo	perfectGas
hhuMixtureThermo	inhomogeneousMixture	constTransport	specieThermo	hConstThermo	perfectGas
hhuMixtureThermo	inhomogeneousMixture	sutherlandTransport	specieThermo	janafThermo	perfectGas
hhuMixtureThermo	veryInhomogeneousMixture	constTransport	specieThermo	hConstThermo	perfectGas
hhuMixtureThermo	veryInhomogeneousMixture	sutherlandTransport	specieThermo	janafThermo	perfectGas

Figure 1.3: Possible entry for thermoType in thermoPhysicalProperties file for the XiFoam solver

class inherited from `hCombustionThermo`, and `hCombustionThermo` inherited from `basicPsiThermo`, and `basicPsiThermo` inherited from `bsicThermo`. So `hhuCombustion` has all the characterestics of `hCombustionThermo` and so on.

⋆ keyword: `stoichiometricAirFuelMassRatio`:
This option is the stoichiometric ratio of Air-Fuel, and is read on line 52 by the following file :

`src/thermophysicalModels/reactionThermo/mixtures/inhomogeneousMixture.C`

Note: `fuel`, `oxidant` and `burntProducts` are also read by this file if in `thermoType` layer `inhomogeneousMixture` was selected.
`reactants`, and `products` coefficient are read on line 52 by the following file:

`src/thermophysicalModels/reactionThermo/mixtures/homogeneousMixture.C`

Full description of these coefficient is in UserGuide[3], as well as the tutorial written by Andreas Lundström[4], but we provide it here again.
The Heat capacity, enthalpy and entropy are evaluated by a function with coefficients from polynomials:

$$\frac{C^{\circ}_{pk}}{R} = a_{1k} + a_{2k}T_k + a_{3k}T_k^2 + a_{4k}T_k^3 + a_{5k}T_k^4 \tag{1.10}$$

$$\frac{H^{\circ}_{k}}{RT_k} = a_{1k} + \frac{a_{2k}}{2}T_k + \frac{a_{3k}}{3}T_k^2 + \frac{a_{4k}}{4}T_k^3 + \frac{a_{5k}}{5}T_k^4 + \frac{a_{6k}}{T_k} \tag{1.11}$$

$$\frac{S^{\circ}_{k}}{R} = a_{1k}\ln T_k + a_{2k}T_k + \frac{a_{3k}}{2}T_k^2 + \frac{a_{4k}}{3}T_k^3 + \frac{a_{5k}}{4}T_k^4 + a_{7k} \tag{1.12}$$

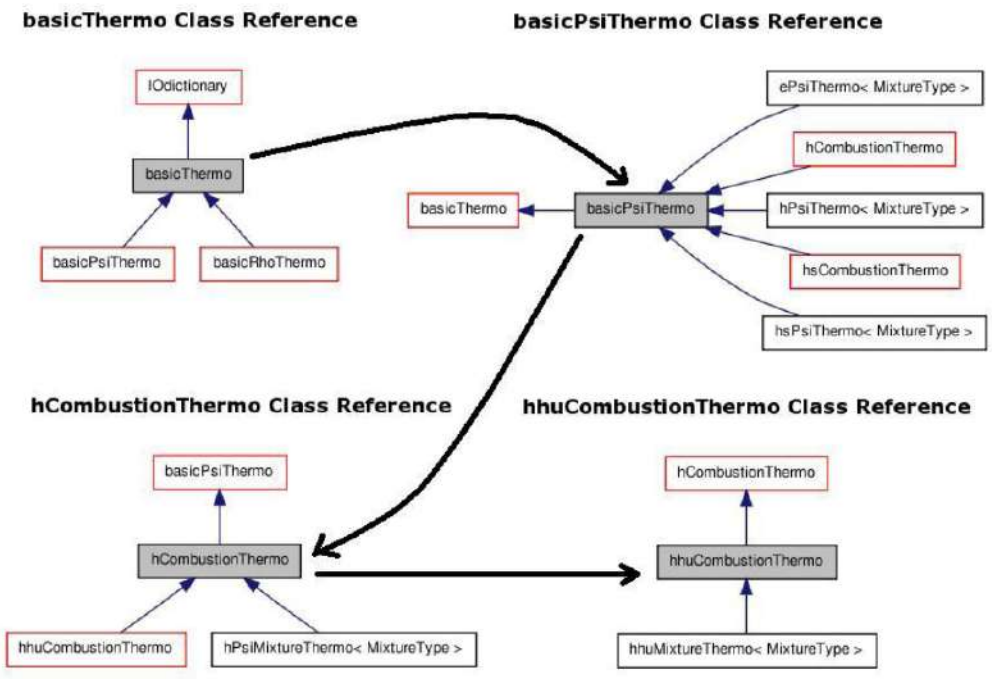


Figure 1.4: Inheritance diagram for hhuMixtureThermo

In order to explain these coefficients , we rewrite them in separate lines as follows:

```
Line 1: fuel
Line 2: fuel 1 44.0962
Line 3: 200 5000 1000
Line 4: 7.53414 0.0188722 -6.27185e-06 9.14756e-10 -4.78381e-14 -16467.5 -17.8923
Line 5: 0.933554 0.0264246 6.10597e-06 -2.19775e-08 9.51493e-12 -13958.5 19.2017
Line 6: 1.67212e-06 170.672;
Line 1: keyword
Line 2: <specieCoeffs>: n_moles Molecular weight(W(kg/kmol))
Line 3: Lower temperature limit Tl(K), Upper temperature limit Th(K), Common temperature Tc(K)
Line 4: High temperature coefficients: a1-a7 (a6:enthalpy offset, a7: entropy offset)
Line 5: Low temperature coefficients: a1-a7 (a6:enthalpy offset, a7: entropy offset)
Line 6: Sutherland coefficient
```

The last two coefficients are Sutherland coefficients which are used to calculate diffusivity as a function of temperature:

$$\mu = A_s \frac{T^{1/2}}{1 + T_s/T} \tag{1.13}$$

The laminar viscosity in turn is calculated using Sutherland's law where the constants are:
$A_s = 1.67212\text{e-}6$
$T_s = 170.672$

1.4.8 combustionProperties

Here is the `combustionProperties` file of the present case. An explanation for each keyword is also presented.

```
laminarFlameSpeedCorrelation Gulders;
fuel            Propane;
Su              Su [ 0 1 -1 0 0 0 0 ] 0.43;
SuModel         unstrained;
equivalenceRatio equivalenceRatio [ 0 0 0 0 0 0 0 ] 1;
sigmaExt        sigmaExt [ 0 0 -1 0 0 0 0 ] 100000;
XiModel         transport;
XiCoef          XiCoef [ 0 0 0 0 0 0 0 ] 0.62;
XiShapeCoef     XiShapeCoef [ 0 0 0 0 0 0 0 ] 1;
uPrimeCoef      uPrimeCoef [ 0 0 0 0 0 0 0 ] 1;
//GuldersEGRCoeffs
GuldersCoeffs
{
    Methane
    {
        W               0.422;
        eta             0.15;
        xi              5.18;
        alpha           2;
        beta            -0.5;
        f               2.3;
    }

    Propane
    {
        W               0.446;
        eta             0.12;
        xi              4.95;
        alpha           1.77;
        beta            -0.2;
        f               2.3;
    }

    IsoOctane
    {
        W               0.4658;
        eta             -0.326;
        xi              4.48;
        alpha           1.56;
        beta            -0.22;
        f               2.3;
    }
}
ignite          yes;
ignitionSites   ( { location ( 0 0 0.0005 ) ; diameter 0.003 ; start 0 ; duration 0.001 ; strength 1 ;
ignitionSphereFraction 1;
ignitionThickness ignitionThickness [ 0 1 0 0 0 0 0 ] 0.001;
ignitionCircleFraction 0.5;
ignitionKernelArea ignitionKernelArea [ 0 2 0 0 0 0 0 ] 0.001;
```

⋆ keyword: `laminarFlameSpeedCorrelation`
There are three different choices for laminar flame speed which presented in figure 1.5:

1- `Gulders`
2- `GuldersEGR`
3- `constant`

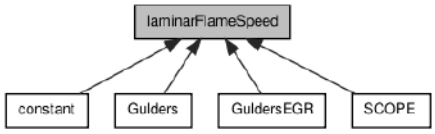
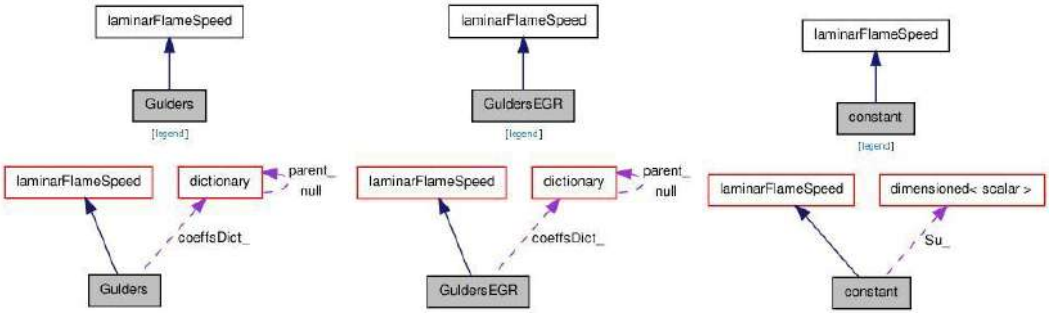


Figure 1.5: laminar flame speed class reference

In the case of selecting `Gulders/GuldersEGR`, laminar flame speed is calculated based on the Gulders formulation. You can see these file for the Gulders formulation:

`src/thermophysicalModels/laminarFlameSpeed/Gulders/Gulders.C`
`src/thermophysicalModels/laminarFlameSpeed/GuldersEGR/GuldersEGR.C`

In these cases(`Gulders,GuldersEGR`), the user must specify the `fuel` and the corresponding coefficients to calculate the laminar flame speed in `GuldersCoeffs/GuldersEGRCoeffs` part, which we will explain in `GuldersCoeffs/GuldersEGRCoeffs` part. Figure 1.6 shows the inheritance diagram for `Gulders`, `GuldersEGR`, and `constant` class. Also, the connection between laminar flame speed class and these classes is presented.



(a) *Gulders* (b) *GuldersEGR* (c) *constant*

Figure 1.6: Gulders, GuldersEGR, and constant class

The diagrams shows that for example `GuldersEGR/Gulders` class gets some coefficient from dictionary(the coefficients that will be discussed later), calculates the laminar flame speed, and returns it to `laminarFlameSpeed` class.

⋆ keyword: `fuel`
The `fuel` name must be specified to calculate the laminar flame speed, if `Gulders/GuldersEGR` is selected in the `laminarFlameSpeedCorrelation` part. `fuel` keyword is read on line 47 of the following file as presented in figure 1.7:

`src/thermophysicalModels/laminarFlameSpeed/laminarFlameSpeed/laminarFlameSpeed.C`

```
line 47: fuel_(dict.lookup("fuel")),
fuel Propane;
```

and it is then used on line 57, 56 of the following files, respectively:

`src/thermophysicalModels/laminarFlameSpeed/Gulders/Gulders.C`
`src/thermophysicalModels/laminarFlameSpeed/GuldersEGR/GuldersEGR.C`

```
line 57, 56: coeffsDict_(dict.subDict(typeName + "Coeffs").subDict(fuel_)),
```

Line 57, 56 specify that based on the `fuel` and `Gulders` or `GuldersEGR` keywords, the necessary coefficients are read from the dictionary. Figure 1.7 shows that keyword `fuel` is read through a dictionary and used in laminar flame speed class.

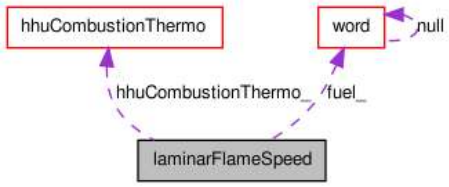


Figure 1.7: Connection between fuel keyword and laminar flame speed class

⋆ keyword: `Su`
If we have the constant laminar flame speed(Su), the `constant` option must be selected on `laminarFlameSpeedCorrelation` and its value must be defined at `Su` keyword.
On line 57 of the following file, constant laminar flame speed(Su) is read.

`src/thermophysicalModels/laminarFlameSpeed/constant/constant.C`

```
line 57 : Su_(dict.lookup("Su"))
Su Su [ 0 1 -1 0 0 0 0 ] 0.43;
```

⋆ keyword: `equivalenceRatio`
EquivalenceRatio of a homogeneous mixture is defined as the ratio of the fuel-to oxidizer ratio to the stoichiometric fuel-to-oxidizer ratio.

$$\phi = \frac{\frac{m_{fuel}}{m_{oxidizer}}}{(\frac{m_{fuel}}{m_{oxidizer}})_{st}} \tag{1.14}$$

This keyword is read by the following file:

`src/thermophysicalModels/laminarFlameSpeed/laminarFlameSpeed/laminarFlameSpeed.C`

⋆ keyword: `SuModel`
There are three options for `SuModel` as follows:

```
1- unstrained
2- equilibrium
3- transport
```

These corresponding keyword are read by the following file:

`applications/solvers/combustion/XiFoam/readCombustionProperties.H`

For more information about the differences between these models, you can refer to line 120 of the following file:

`applications/solvers/combustion/XiFoam/bEqn.H`

⋆ keyword: `sigmaExt`
`sigmaExt` is the strain rate at extinction which obtained from the Markstein length by extrapolating to S_u --> 0 .
Markstein length: Length that measures the effect of curvature on a flame. The larger the Markstein length, the greater the effect of curvature on burning velocity. The Markstein length divided by the flame thickness is the Markstein number.
This keyword is read by the following file:

`applications/solvers/combustion/XiFoam/readCombustionProperties.H`

and used in `myBEqn.H` file.

⋆ keyword: `XiModel`
There are different models for flame wrinkling Xi=St/Sl (turbulent flame speed/laminar flame speed) which are:

```
1- fixed
2- algebraic
3- transport
```

These keywords are read by the following file:

`applications/solvers/combustion/XiFoam/readCombustionProperties.H`

For more information about the differences between implementation of these models, you can refer to line 166 of the following file:

`applications/solvers/combustion/XiFoam/bEqn.H`

And also for more details regarding the formulation you can refer to Weller Article[2].

⋆ keyword: `XiCoef and XiShapeCoef`
`XiCoef` and `XiShapeCoef` are used in the `algebraic` model for Xi on line 175 of bEqn.H which has already been presented in section 1.3.3 .
These keywords are read by the following file:

`applications/solvers/combustion/XiFoam/readCombustionProperties.H`

⋆ keyword: `uPrimeCoef`
`uPrimeCoef` is used for calculation of the velocity fluctuation on line 74 of the `bEqn.H`:

```
line 74: volScalarField up = uPrimeCoef*sqrt((2.0/3.0)*turbulence->k());
```

⋆ keyword: `GuldersCoeffs/ GuldersEGRCoeffs`:
Here are the coefficients, which are used to calculate laminar flame speed according to the Gulders formulation for specific fuel. As we told, these coefficients are read by the following codes depending on the model that has been selected for `laminarFlameSpeedCorrelation`.

`src/thermophysicalModels/laminarFlameSpeed/Gulders/Gulders.C`
`src/thermophysicalModels/laminarFlameSpeed/GuldersEGR/GuldersEGR.C`

So if any one would like to modify the laminar flame speed models, he/she probably should modify these codes. Here is the formulation for calculating the laminar flame velocity based on `GuldersCoeffs` `GuldersEGRCoeffs` coefficients:

$$S_u = W\Phi^{\eta} exp[-\xi(\Phi - 1.075)^2](\frac{T}{T_0})^{\alpha}(\frac{P}{P_0})^{\beta} \tag{1.15}$$

where, Φ is a equivalence ratio.

⋆ keyword: `ignite`
If you have ignition, it must be specified here. This entry is read by the `readCombustionProperties.H` file on line 45:

```
line 45: ignition ign(combustionProperties, runTime, mesh);
```

⋆ keyword: `ignitionSites`
The `location` of ignition, the `duration` and the `strength` are specified at `ignitionSites` keyword as follows:

```
ignitionSites ({location (0 0 0.0005); diameter 0.003; start 0; duration 0.001; strength 1;});
```

These data are read by the following code:
`src/engine/ignition/ignitionSiteIO.C`

⋆ keyword: `ignitionSphereFraction, ignitionThickness, ignitionCircleFraction, ignitionKernelArea`
These factors are read by the following files:
`src/engine/include/stCorr.H`
`StCorr` is used when calculating the turbulent flame speed flux in bEqn.H file on line 37:

```
line 37: surfaceScalarField phiSt = fvc::interpolate(rhou*StCorr*Su*Xi)*nf;
```

`StCorr` varies between 1-10 during the simulation.
`StCorr.H` file uses `mesh.nGeometricD()` function which checks the shape of the mesh, the number of valid geometric dimensions in the mesh, and returns the value 3, 2 or 1 which correspond to:

3: Assume it is part-spherical
2: Assume it is part-circular
1: Assume it is plane or two planes

Depending on the geometry (or in other words, depends on these values(3, 2, or 1) which return by `NgeometricD()`), one of the above factors (`ignitionSphereFraction`, `ignitionThickness`, `ignitionCircleFraction`, `ignitionKernelArea`) is used.

1.5 /0 Folder

We have the following files in `0` directory, which must be modified based on the new geometry and boundary conditions.

```
alphat b epsilon k mut p Su T Tu U Xi
```

1.5.1 Boundary Conditions

Boundary conditions for all sides are symmetry planes.

```
boundaryField
{
    left
    {
        type            symmetryPlane;
    }
    right
    {
        type            symmetryPlane;
    }
    top
    {
        type            symmetryPlane;
    }
    bottom
    {
        type            symmetryPlane;
    }
    front
    {
        type            symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }
}
```

1.5.2 Initial Conditions

The initial condition are found in `/0` directory. Table 1.2 presents the initial conditions. Since there is no ignition at time = 0, we have `b`=1 and `Xi`=1 in the whole domain.

Variable	Description	Initial Condition
alphat	Turbulence thermal diffusivity [kg/m/s]	internalField uniform 0
b	Regress variable (dimensionless)	internalField uniform 1
epsilon	The turbulence kinetic energy dissipation rate $[m^2/s^3]$	internalField uniform 375
k	the turbulence kinetic energy $[m^2/s^2]$	internalField uniform 1.5
mut	the turbulence viscosity [kg/m/s]	internalField uniform 0
p	Pressure $[kg/m/s^2]$	internalField uniform 100000
S_u	Laminar flame speed [m/s]	internalField uniform 0.43
T	Temperature [K]	internalField uniform 360
T_u	Unburnt Temperature [K]	internalField uniform 360
U	Velocity Field [m/s]	internalField uniform (0 0 0)
Xi	The flame-wrinking S_t/S_u(dimensionless)	internalField uniform 1

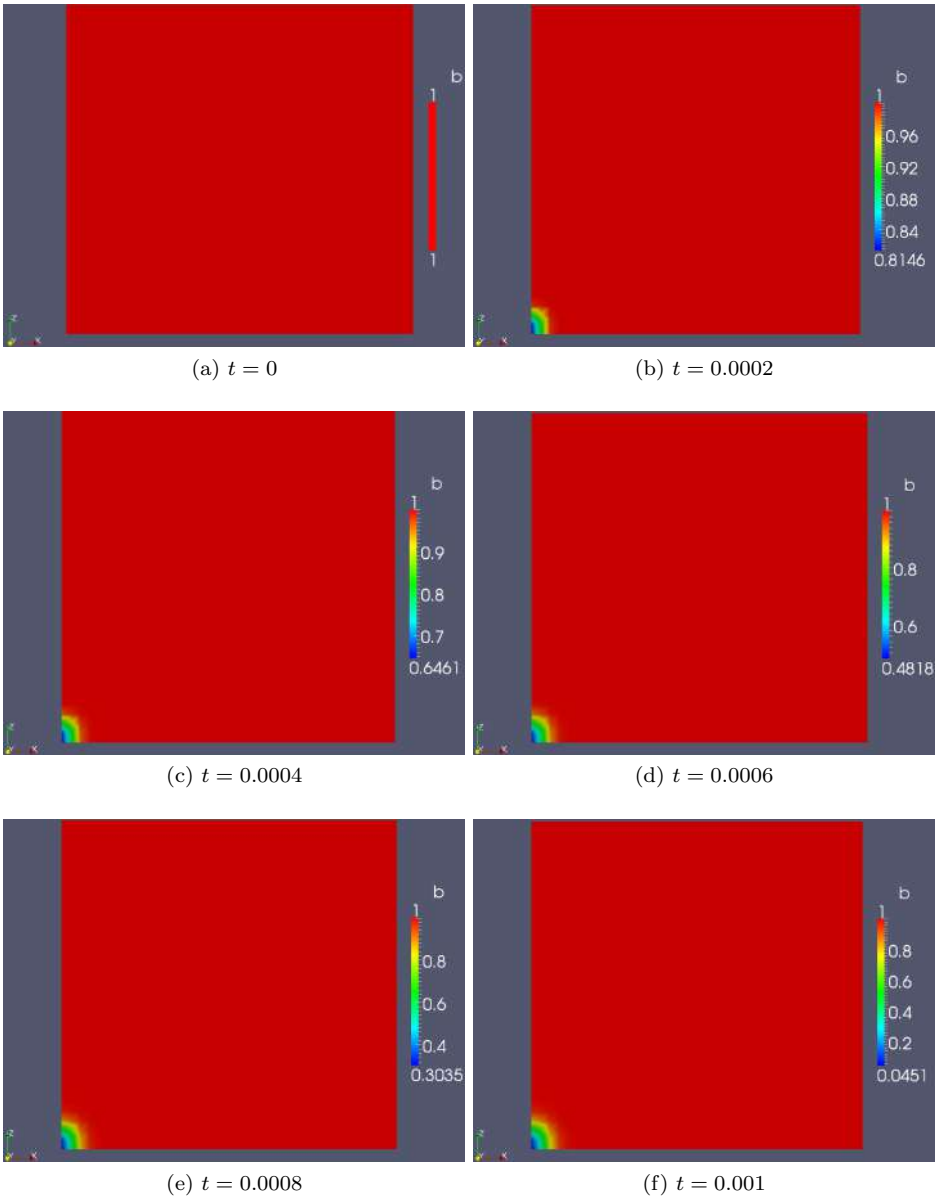
Table 1.2: Initial Condition

1.6 System

There is no need to change the system directory files.
Now you can run the simulation:

```
myXiFoam >log &
paraFoam
```

Figure 1.8 shows the regress variable at several time steps. As it can been seen, the flame propagates through unburned gas. The value of `b` is equal to 1 at start time, and then it decreases and at the ignition time it has its minimum value.



(a) $t = 0$ (b) $t = 0.0002$

(c) $t = 0.0004$ (d) $t = 0.0006$

(e) $t = 0.0008$ (f) $t = 0.001$

Figure 1.8: The variation of the regress variable at different time steps

1.7 Implementing a new algebraic combustion model for Xi

In this section we are going to implement a new algebraic model to calculate the turbulent flame speed. For details explanation of this model please refer to the Lipatnikov paper [5].
Equation 1.16 presnts a new algebraic model for calculation the turbulent flame speed.
Nore: All files which are created here must be in the same directory as the solver directory.

$$S_{turbTimeDependent} = S_{turb}[1 + \frac{\tau_t^{'}}{t + t_0}(exp(\frac{-t + t_0}{\tau_t^{'}}))]^{\frac{1}{2}} \tag{1.16}$$

where:

$$S_{turb} = XiShapeCoeffDa^{0.25}u' \tag{1.17}$$

`Da` is Damköhler number.

$$Da = \frac{\tau_t}{\tau_c} \tag{1.18}$$

$$t_0 = 0.1\tau_t \tag{1.19}$$

τ_t and $\tau_t^{'}$ are the turbulent time scales.

$$\tau_t = \frac{L}{u'} \tag{1.20}$$

$$\tau_t^{'} = \frac{\alpha_{turb}}{u'^2} \tag{1.21}$$

τ_c is the chemical time scale.

$$\tau_c = \frac{\alpha_{molecular}}{S_{L,0}^2} \tag{1.22}$$

α_{turb} and $\alpha_{molecular}$ are turbulent and molecular thermal diffusivity, respectively. `L` is the turbulence length scale.

$$L = C_\mu^{0.75}\frac{k^{3/2}}{\epsilon} \tag{1.23}$$

k is the turbulent kinetic energy, and ϵ is the turbulent dissipation rate.

To implement this model, it is necessary to create the required fields related to the equations 1.18 to 1.16 and initialized them. This is done by creating the `myCreateFields.H` file as follows:

```
gedit myCreateFields.H
```

And write the following lines, which create the required fileds and initialized them:

```
Info<<"Reading myCreateFields.H File"<<endl;

scalar Cmu=0.09;
volScalarField up = uPrimeCoef*sqrt((2./3.)*turbulence->k());
volScalarField L=Cmu*pow(turbulence->k(),1.5)/turbulence->epsilon();

volScalarField tauTurb = L/up;
volScalarField tauTurbPrime = (turbulence->alphaEff()-thermo.alpha())/(rho*pow(up,2));
volScalarField tauChem = thermo.alpha()/(rho*pow(up,2));
volScalarField Da = tauTurb/tauChem ;
volScalarField t0=0.1*L/up;
Info<< "Calculating turbulent flame speed field S_turb\n" << endl;
volScalarField S_turb
(
```

```
        IOobject
        (
            "S_turb",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        XiShapeCoef*up*pow(Da,0.25)
    );

    Info<< "Calculating turbulent flame speed field S_turbTimeDependent \n" << endl;
    volScalarField S_turbTimeDependent
    (
        IOobject
        (
            "S_turbTimeDependent",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        S_turb*sqrt(1.+tauTurbPrime/(runTime+t0)*(exp(-1.*(runTime+t0)/tauTurbPrime)-1.))
    );
```

Moreover, these fields must be calculated during the simualtion. Therefore, creates `newStModel.H` and defines parameters in the equations 1.18 to 1.16:

```
gedit newStModel.H
```

And write:

```
Info<< "Reading newStModel.H file \n"<< endl;

L = Foam::pow(Cmu,scalar(0.75))*pow(turbulence->k(),1.5)/turbulence->epsilon();
tauTurb = L/up;
tauTurbPrime = (turbulence->alphaEff()-thermo.alpha())/(rho*pow(up,2));
tauChem = thermo.alpha()/(rho*pow(up,2));
Da = tauTurb/tauChem ;
t0=0.1*L/up;

S_turb = XiShapeCoef*up*pow(Da,0.25);
S_turbTimeDependent=S_turb*
                    sqrt(1.+tauTurbPrime/(runTime+t0)*(exp(-1.*(runTime+t0)/tauTurbPrime)-1.));
Xi=S_turbTimeDependent/Su;
```

Finally, these file must be included in the appropriate files. So, some modification in `myBEqn.H` and `myXiFoam.H` must be done.
In myXiFoam.H after the `#include "createFields.H"` add:

```
#include "myCreateFields.H"
```

And in `myBEqn.H` we must include the new algebraic model. After :

```
if (XiModel == "fixed")
{
    // Do nothing, Xi is fixed!
}
```

Add:

```
else if (XiModel == "newAlgebraic")
{
    #include    "newStModel.H"
}
```

Then run

```
wmake
```

Now `myXiFoam` solver with new combustion model is ready to run. In order to use this solver with new combustion model, the user must specified `newAlgebraic` for the `XiModel` keyword in `combustionProperties` file.

Bibliography

[1] Fluent help document.(http://jullio.pe.kr/fluent6.1/help/html/ug/node574.htm)

[2] H. G. Weller, G. Tabor, A. D. Gosman and C. Fureby. 1998. Application of a Flame-Wrinkling LES Combustion Model to a Turbulent Mixing Layer *Twenty-Seventh Symposium (International) on Combustion/The Combustion Institute*, pp. 899–907

[3] OpenFOAM UserGuide for Version 1.7

[4] OpenFOAM Tutorial, reactingFoam, Andreas Lundstöm, Chalmers University of Technology, April 2008, OpenFOAM Course.http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2007/AndreasLundstrom/reactingFoam.pdf

[5] A. N. Lipatnikov and J. Chomiak. Turbulent flame speed and thickness: phenomenology, evaluation, and application in multi-dimensional simulations *Progress in Energy and Combustion Science*, Volume 28, Issue 1, 2002, Pages 1-74