Cite as: Zhang, M.: Implementation of a two-equation soot model for sprayFoam. In Proceedings of CFD with OpenSource Software, 2019, Edited by Nilsson. H., http://dx.doi.org/10.17196/OS_CFD#YEAR_2019

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of a two-equation soot model for sprayFoam

Developed for OpenFOAM-v1712

Author:
Min Zhang
Technical University of Denmark
mzhang@mek.dtu.dk

Peer reviewed by:

Luis Fernando Garcia Rodriguez

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 14, 2020

Learning outcomes

The main requirements of a tutorial 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 use the sprayFoam to run the AachenBomb case
- How to set basic spray parameters including the breakup model and injector parameters
- How to set chemkin file

The theory of it:

- Basic knowledge of soot formation and oxidation
- Two transport equations for soot model will be discussed

How it is implemented:

- A brief overview for how the sprayFoam links the soot model included in thermophysicalModel library
- A basic understanding for existed mixtureFractionSoot model to see how it calculate soot using the single chemical reaction.

How to modify it:

- The existed mixtureFractionSoot model is only used for the single chemical reaction. In fact, the soot formation is affected by a great range of species including C2H2, OH and O2, also by turbulence.
- To make the soot model take the turbulence and key species into account, the newly soot model will be developed by adding the two transport equations. These two transport equations govern the distribution of soot
- The source terms in the transport equations need to be modeled by soot modelling.
- Corresponding species are needed read from thermophysicalModel library
- Corresponding fields for soot model will be created
- Corresponding soot constants should be set from soot properties in constant fold

Prerequisites

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

- Basic theory of combustion
- Run standard document tutorials like sprayFoam tutorial
- It is strongly recommended to gain a brief insight into soot formation and oxidation from the following journal [4] and PhD thesis [3](if accessible):

  Pang, K.M., Jangi, M., Bai, X.S. and Schramm, J., 2015. Evaluation and optimisation of phenomenological multi-step soot model for spray combustion under diesel engine-like operating conditions. Combustion Theory and Modelling, 19(3), pp.279-308.
  Ong, J.C., 2017. Development of Lagrangian soot tracking method for the study of soot morphology in diesel spray combustion (Doctoral dissertation, University of Nottingham).

Contents

Chapter 1

Introduction

The spray combustion event is recognized as one of the most complex in-cylinder phenomena to understand and control. This event refers to fuel droplet atomisation, mixture preparation, ignition, combustion and soot production. Soot particle is an important source of PM(particulate matter)2.5, which causes severely healthy problem. Hence, investigation on soot production is an effective way to reduce the emissions from the engine.

In OpenFoam-v1712, only one soot model called mixtureFractionSoot has been implemented. This soot model is a simple state model. It does not solve the transport equations which control the soot distribution. Also, this model calculate the soot mass fraction based on the CO_2 mass fraction instead of considering the soot formation and oxidation. Soot production is a extremely physical and chemical process which refers to soot inception, coagulation, surface growth, oxidation by OH and O_2. Hence, implementing a multistep soot model and transport equations to OpenFOAM is necessary.

Chapter 2

Theory

2.1 Soot formation and oxidation

A successful modelling of soot requires accurate accounting for both formation and oxidation of soot. Furthermore, it is important that the soot models developed and employed are able to give good predictions of soot information and oxidation. Models of different sophistication levels have been proposed and applied in many practical systems, and can be generally classified into below four categories:

- Empirical model: It uses correlations of experimental data to predict trends in soot production. They are implemented and provide excellent correlations for a given set of operating conditions. Obviously, the empirical soot model can not predict well when the operating conditions are changed a lot compared to the given operating conditions.
- Semi-empirical model: It solves rate equations that are calibrated using experimental data
- Detailed model: It contains full panoply of soot phenomena, from the initial pyrolysis of fuel to the inception of soot particles, surface growth, coagulation and oxidation. However, such comprehensive models impose heavy computational cost.
- Phenomenological multistep soot model: Phenomenological multistep models use sub-models developed to describe the different processes. These sub-models can be empirically developed by using basic physical and chemical relations. Basically, the global reaction steps can be used to describe the every submodel.

As mentioned earlier, the phenomenological multistep models have advantages in both cheap computation and good computational accuracy. Figure 2.1 shows the basic process of multistep soot model.

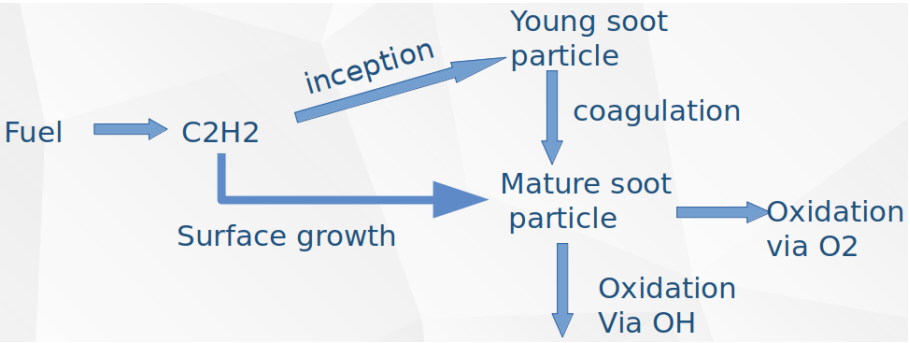

Figure 2.1: phenomenological multistep models process.

From the Figure 2.1, we can see the soot model consists of inception, coagulation, surface growth, oxidation by OH and O_2. Particle inception can be considered to be a process of physical condensation or a process of chemical (reactive) condensation. The coagulation and agglomeration process are physical processes. Coagulation,sometimes called coalescence, is the process where two or more particles collide and combine to form a larger particle. Surface growth is the process of adding mass to the surface of a nucleated soot particle. Soot particle oxidation is the mechanism by which soot particles is oxidised and converted back into gaseous species. Just like surface growth, oxidation is a heterogeneous process that takes place on the particle surface and occurs throughout the entire course of the soot formation. These submodels with their respective descriptions and model constant values are summarized in table 2.1

i	Description	C	b	T_a	n
inc	Inception	10000	0	21000	0
sg	Surface growth	$(p/p_{ref})^{1.4}$	0	12100	0.5
OH	Oxidation due to OH	0.36	0.5	0	1
O_2	Oxidation due to O_2	10000	0.5	19778	1

Table 2.1: The multistep soot model constant values

The reaction rate of each sub-process is calculated using the Arrhenius expression,

$$k_i = C_i T^{b,i} \cdot exp\left(\frac{-T_{a,i}}{T}\right)(S_{soot})^{n,i} \tag{2.1}$$

T in equation 2.1 represents the gas temperature, C_i here denotes the model constant, while $T_{a,i}$ is the activation temperature. It should be noted that S_{soot} is the soot specific area in the soot model. From the formulation, we can see that the inception rate is independent of this parameter while the surface growth rate and oxidation rate are a square root function and a linear function of this parameter, respectively. When we describe the soot distribution, the most important two parameters are the soot mass fraction, M_{soot}, and particle number density, N_{soot}. The source term for the M_{soot} transport equation denotes the net soot mass production and is modelled according to the expression,

$$\frac{dM_{soot}}{dt} = k_{inc}[X_{prec}] + k_{sgs}[C_2H_2] - k_{OH}[OH] - k_{O_2}[O_2] \tag{2.2}$$

where M_{soot} denotes the soot mass concentration.
For the particle number density, N_{soot}, it is the instantaneous production rate of soot particles, which is subjected to inception from the gas phase and coagulation in the free molecular regime and it is computed using equation 2.3,

$$\frac{dN_{soot}}{dt} = N_a k_{inc} - k_{coag} \tag{2.3}$$

where N_a is the Avogadro constant, k_{inc} is the inception rate, k_{coag} is the coagulation rate and described as below[4],

$$k_{cog} = k_{cog}(\frac{24RT}{\rho_{soot}N_A})^{1/2}(\frac{6M_{soot}}{\rho_{soot}\pi})^{1/6}(N_{soot})^{11/6} \tag{2.4}$$

2.2 Transport equations

The well-know transport equations are Moss-Brookes (MB) two transport equations. These two transport equations are computed via Eulerian method and able to account for individual soot processes and turbulence effect. Below these two equations are given as,

$$\frac{\partial}{\partial t}(\rho Y_{soot}) + \nabla \cdot (\rho \mathbf{U} Y_{soot}) = \nabla \cdot [\frac{\mu_t}{Pr_{soot}} \nabla Y_{soot}] + \frac{dM_{soot}}{dt} \tag{2.5}$$

where rho is density, U is flow speed, Y_{soot} is soot mass fraction and μ_t is the effective turbulent viscosity.

$$\frac{\partial}{\partial t}(\rho b^*_{nuc}) + \nabla \cdot (\rho \mathbf{U} b^*_{nuc}) = \nabla \cdot [\frac{\mu_t}{Pr_{nuc}} \nabla b^*_{nuc}] + \frac{1}{N_{normal}} \frac{dN_{soot}}{dt} \tag{2.6}$$

where b^*_{nuc} is the normalised radical nuclei concentration, the turbulent Prandtl number for soot transport and nuclei transport is represented by Pr_{soot} and Pr_{nuc}, respectively. N_{normal} is a normalisation factor with a value of 10^{15}particles. The source term for the soot mass fraction, Y_{soot} in equation 2.5 computes the production of soot mass and is expressed as equation 2.2. The source term for the normalised radical nuclei concentration, b^*_{nuc} in equation 2.6 is given in equation 2.3.

Chapter 3

Implementation to a radiation library

3.1 The creation of myRadiation library

The soot model is invoked by radiation model and the radiation model is created by:

```
#include "createRadiationModel.H"
```

Therefore, we need to copy the radiation library into the user project folder `$WM_PROJECT_USER_DIR` without any changes.

```
of1712
foam
cp -r --parents src/thermophysicalModels/radiation $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/
```

Rename the library name by following command,

```
mv radiation myRadiation
```

Modify the executable name to `libmyradiationModels.so` by conducting the follow command,

```
cd myRadiation
sed -i  s/libradiationModels/libmyradiationModels/g Make/files
```

In order to make the executable file "libmyradiationModels.so" locate at the `$FOAM_USER_LIBBIN` instead of `$FOAM_LIBBIN`, we need to conduct following command,

```
sed -i  s/FOAM_LIBBIN/FOAM_USER_LIBBIN/g Make/files
```

Compile it after changing,

```
wmake
```

If successful, you will see the output at the end like "libmyradiationModels.so"

3.2 The understanding of file structure for mixtureFractionSoot model

Let's go into mixtureFractionSoot model by following commands,

```
cd submodels/sootModel/mixtureFractionSoot
```

The `mixtureFractionSoot` consists of three files. The `mixtureFractionSoots.C` file is responsible for making a new type for `mixtureFractionSoot`. The `mixtureFractionSoot.H` is used to declare the member data and functions. The `mixtureFractionSoot.C` is used to initialize the data member and define the member function. The soot mass fraction is calculated by the member function `correct()`. The definition of `correct()` is given as bellow,

```
const volScalarField& mapField =
mesh_.lookupObject<volScalarField>(mappingFieldName_);
soot_ = sootMax_*(mapField/mapFieldMax_);
```

If you want to see more detailed tutorials about the mixtureFractionSoot model, you can refer to the [6]

3.3 The modification of multistep soot model

As mentioned before, there is only mixtureFractionSoot model existed in OpenFOAM. We need to do the modification based on this soot model. Hence, copy the folder and rename it.

```
cd ..
cp -r mixtureFractionSoot multistepSoot
```

Also, we need to rename the files inside the folder,

```
cd multistepSoot
mv mixtureFractionSoot.C multistepSoot.C
mv mixtureFractionSoot.H multistepSoot.H
mv mixtureFractionSoots.C multistepSoots.C
```

All these three files need to be modified, the multistepSoots.C is used to change the soot type from "mixtureFractionSoot" to "multistepSoot" by using the below commands,

```
sed -i s/mixtureFractionSoot/multistepSoot/g multistepSoots.C
```

For the multistepSoot.H, we also need to change the `mixtureFractionSoot` to `multistepSoot` by using following command,

```
sed -i s/mixtureFractionSoot/multistepSoot/g multistepSoot.H
```

In order to solve a couple of chemistry reactions, we have to change the `singleStepReactingMixture.H` to `reactingMixture.H` by following command,

```
sed -i s/singleStepReactingMixture/reactingMixture/g multistepSoot.H
```

In order to consider the turbulent effect on soot distribution and read the effective turbulent viscosity from turbulence model, the `turbulentFluidThermoModel.H` should be included after the `reactingMixture.H`.
All the declarations are written in multistepSoot.H, considering to add the multistep soot model and two transport equations into the multistepSoot model, the following declarations should be added under the //Private data,

```
volScalarField PDSoot_;
volScalarField dNSootdt_;
volScalarField SootVF_;
volScalarField SSoot_;
volScalarField dMSootdt_;
volScalarField r1_;
volScalarField r2_;
volScalarField r3_;
```

```
volScalarField r4_;
volScalarField r5_;
volScalarField MSoot_;
volScalarField NSoot_;

scalar Sct_;
scalar MWc_;
scalar wPrec_;
scalar rhoS_;
scalar Ca_;
scalar Ta_;
scalar Cb_;
scalar Cg_;
scalar Tg_;
scalar m_;
scalar q_;
scalar Cw1_;
scalar CollEff_;
scalar Cw2_;
scalar Tw2_;
scalar NA_;
```

It should be noted that `PDSoot_` is the soot particle density, `dNSootdt_` is the soot particle changing rate, `YSoot_` is the soot mass fraction, `SootVF_` is soot volume fraction, `SSoot_` is the soot surface area, `dMSootdt_` is the soot mass changing rate, `r1_` is the inception rate, `r2_` is the coagulation rate, `r3_` is the surface growth rate, `r4_` is the soot oxidation rate by `OH`, r5 is the soot oxidation rate by `O2`, `MSoot_` is soot mass, `NSoot_` is the soot particle number.

There are some declarations including `nuSoot_`, `Wsoot_`,`sootMax_`, `mappingFieldName_`, `mapFieldMax_` and `mixture_` for mixtureFractioSoot model we do not need them any more, we just delete them.

The final file we need to modify is `multistepSoot.C`, we should define the data member and member functions here. Firstly, we need to change the `mixtureFractionSoot` to the `multistepSoot` by using sed command,

```
sed -i s/mixtureFractionSoot/multistepSoot/g multistepSoot.C
```

Similar to the `multistepSoot.H` file, we also need to change the `singleStepReactingMixture.H` to `reactingMixture.H`,

```
sed -i s/singleStepReactingMixture/reactingMixture/g multistepSoot.C
```

We need to add two transport equations to this model, following header file should be included,

```
#include "fvmDdt.H"
#include "fvmDiv.H"
#include "fvcDiv.H"
#include "fvmLaplacian.H"
```

As we do not need to use the way in mixtureFractionModel to calculate the soot, we can delete all the unnecessary codes. First, we delete the codes after the `coeffsDict_(dict...),` and before the `thermo_(mesh...),`. Meanwhile, we add the follow codes between these two aforementioned line codes to to initialize the data member.

```
Sct_(readScalar(coeffsDict_.lookup("Sct"))),
MWc_(readScalar(coeffsDict_.lookup("MWc"))),
```

```
wPrec_(readScalar(coeffsDict_.lookup("wPrec"))),
rhoS_(readScalar(coeffsDict_.lookup("rhoS"))),
Ca_(readScalar(coeffsDict_.lookup("Ca"))),
Ta_(readScalar(coeffsDict_.lookup("Ta"))),
Cb_(readScalar(coeffsDict_.lookup("Cb"))),
Cg_(readScalar(coeffsDict_.lookup("Cg"))),
Tg_(readScalar(coeffsDict_.lookup("Tg"))),
m_(readScalar(coeffsDict_.lookup("m"))),
q_(readScalar(coeffsDict_.lookup("q"))),
Cw1_(readScalar(coeffsDict_.lookup("Cw1"))),
CollEff_(readScalar(coeffsDict_.lookup("CollEff"))),
Cw2_(readScalar(coeffsDict_.lookup("Cw2"))),
Tw2_(readScalar(coeffsDict_.lookup("Tw2"))),
NA_(readScalar(coeffsDict_.lookup("NA"))),
```

Then, we also do not need to have `mixture_(checkThermo(thermo_))`, it should be deleted. Meanwhile, note that the comma `,` after the `thermo_(mesh...),` should be deleted.
To create the the relevant fields for multistep soot model, before the `coeffsDict_(dict...),`, we add the following codes,

```
    PDSoot_
    (
        IOobject
        (
            "PDSoot",
            mesh_.time().timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),

    dNSootdt_
    (
        IOobject
        (
            "dNSootdt",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("dNSootdt", dimensionSet(0,-3,-1,0,0,0,0), 0.0)
    ),

    SootVF_
    (
        IOobject
        (
            "SootVF",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
```

```
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("dNSootdt", dimensionSet(0,-3,-1,0,0,0,0), 0.0)
    ),

    SSoot_
    (
        IOobject
        (
            "SSoot",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("SSoot", dimensionSet(0,-1,0,0,0,0,0), 0.0)
    ),

    dMSootdt_
    (
        IOobject
        (
            "dMSootdt",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("dMSootdt", dimensionSet(1,-3,-1,0,0,0,0), 0.0)
    ),

    r1_
    (
        IOobject
        (
            "r1",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("r1", dimensionSet(0,-3,-1,0,1,0,0), 0.0)
    ),

    r2_
    (
        IOobject
        (
            "r2",
            mesh_.time().timeName(),
```

```
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("r2",dimensionSet(0,-3,-1,0,0,0,0), 0.0)
    ),

    r3_
    (
        IOobject
        (
            "r3",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("r3", dimensionSet(0,-3,-1,0,1,0,0), 0.0)
    ),

    r4_
    (
        IOobject
        (
            "r4",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("r4", dimensionSet(0,-3,-1,0,1,0,0), 0.0)
    ),

    r5_
    (
        IOobject
        (
            "r5",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("r5", dimensionSet(0,-3,-1,0,1,0,0), 0.0)
    ),

    MSoot_
    (
        IOobject
        (
```

```
            "MSoot",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("MSoot", dimensionSet(1,-3,0,0,0,0,0), 0.0)
    ),

    NSoot_
    (
        IOobject
        (
            "NSoot",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("NSoot", dimensionSet(0,-3,0,0,0,0,0), 0.0)
    ),

    YPrec_
    (
        IOobject
        (
            "YPrec",
            mesh_.time().timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("YPrec", dimensionSet(0,0,0,0,0,0,0), 0.0)
    ),
```

It should be noted that the `PDSoot` and `soot` must read from 0 folder.

For the constructor we delete all the code in the `{ }`.For the Member Functions we also need to delete the codes in the `{ }` and add the new code inside. Following codes are added with the order we give in this tutorial.

To read the flux, density, temperature and pressure, we use the following codes,

```
const surfaceScalarField& phi_(mesh_.lookupObject<surfaceScalarField>("phi"));
const volScalarField& rho_(mesh_.lookupObject<volScalarField>("rho"));
const volScalarField& T = thermo_.T();
const volScalarField& p = thermo_.p();
```

Some chemistry constants are given as follows,

```
scalar kb = 1.3806488e-23;
scalar Pref = 101325;
```

```
scalar wC2H2 = 26.03824;
scalar wOH = 17.00737;
scalar wO2 = 31.9988;
```

In the multistep soot model, some key species concentration including `OH` and `O2` should be obtained to calculate the submodel rate. The following codes are included to look up the concentration,

```
const volScalarField& Y_C2H2 = mesh_.lookupObject<volScalarField>("C2H2");
const volScalarField& Y_O2 = mesh_.lookupObject<volScalarField>("O2");
```

The following codes are added to calculate the submodel rate in equation 2.1,

```
forAll(T, celli)
{
scalar Ti = T[celli];
if (soot_[celli]<1.0e-12)
{
  MSoot_[celli] = 0.0;
}
else
{
MSoot_[celli] = soot_[celli]*rho_[celli];
}
if (PDSoot_[celli]<1.0e-15)
{
  NSoot_[celli] = 0.0;
}
else
{
NSoot_[celli] = PDSoot_[celli]*rho_[celli]*NA_;
}

SootVF_[celli] = soot_[celli]*rho_[celli]/rhoS_;
SSoot_[celli] = pow(6*MSoot_[celli]/rhoS_,2.0/3.0)*pow(constant::mathematical::
pi*NSoot_[celli],1.0/3.0);
r1_[celli] = Ca_*(Y_C2H2[celli]*rho_[celli]/wPrec_)*exp(-Ta_/Ti);
r2_[celli] = Cb_*sqrt(24*kb*T[celli]/rhoS_)*pow(6*MSoot_[celli]/constant::mathematical::
pi/rhoS_,1.0/6.0)*pow(NSoot_[celli],11.0/6.0);
if (m_ == 0.5) //
{
  r3_[celli] = pow(p[celli]/Pref,q_)*Cg_*(Y_C2H2[celli]*
  rho_[celli]/wC2H2)*exp(-Tg_/Ti)*sqrt(SSoot_[celli]);
}
else if (m_ == 1.0)
{
  r3_[celli] = pow(p[celli]/Pref,q_)*Cg_*(Y_C2H2[celli]*
  rho_[celli]/wC2H2)*exp(-Tg_/Ti)*SSoot_[celli];
}
else
{
FatalErrorIn
         (
             ""
         )   << "Invalid integer value for m" << nl
            << "Valid integer values are 0.5 and 1.0" << nl
                  << "m = 0.5 indicates surface growth rate is a
```

```
                square root function of SSoot" << nl
                << "m = 1.0 indicates surface growth rate is a
                linear function SSoot" << nl
            << exit(FatalError);
}
if (Cw1_ > 0)
{
const volScalarField& Y_OH = mesh_.lookupObject<volScalarField>("OH");
r4_[celli] = Cw1_*CollEff_*(Y_OH[celli]*rho_[celli]/wOH)*sqrt(Ti)*SSoot_[celli];
}
r5_[celli] = Cw2_*(Y_O2[celli]*rho_[celli]/wO2)*exp(-Tw2_/Ti)*sqrt(Ti)*SSoot_[celli];
dNSootdt_[celli] = NA_*r1_[celli]-r2_[celli];
dMSootdt_[celli] = MWc_*(100*r1_[celli]+2*r3_[celli]-r4_[celli]-r5_[celli]);
scalar deltaT =0.0;
deltaT = this -> mesh().time().deltaTValue();
if (MSoot_[celli]+dMSootdt_[celli]*deltaT<0.0)
dMSootdt_[celli]=-MSoot_[celli]/deltaT;
if (NSoot_[celli]+dNSootdt_[celli]*deltaT<0.0)
dNSootdt_[celli]=-NSoot_[celli]/deltaT;
}
```

In the two transport equations, there is a term called effective eddy viscosity, we have to read this from turbulence model using the following commands,

```
const compressible::turbulenceModel&
turbModel = rho_.db().lookupObject<compressible::turbulenceModel>
(
    turbulenceModel::propertiesName
);
```

To solve the two transport 2.5 and 2.6, the following codes need to be added,

```
   tmp<fvScalarMatrix> PDSootEqn
  (
        fvm::ddt(rho_, PDSoot_)
        +fvm::div(phi_, PDSoot_)
        -fvm::laplacian(turbModel.muEff()/Sct_, PDSoot_)
      ==
        dNSootdt_/(NA_*dimensionedScalar("unity",dimless/dimMoles,1.0))
   );
    solve(PDSootEqn);
   tmp<fvScalarMatrix> sootEqn
   (
         fvm::ddt(rho_, soot_)
         +fvm::div(phi_,soot_)
         -fvm::laplacian(turbModel.muEff()/Sct_, soot_)
       ==
         dMSootdt_
    );
     solve(sootEqn);
```

Before we compile this model, we need to add this model to `Make/files` in `myRadiation`,

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/myRadiation
vi Make/files
```

Add following code after `submodels/sootModel/mixtureFractionSoot/mixtureFractionSoots.C`,

`submodels/sootModel/multistepSoot/multistepSoots.C`

Also, we need to link the turbulence library to `Make/options`, under the `EXE_INC = \`, add

```
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
```

under the `LIB_LIBS = \`, add

```
    -lcompressibleTurbulenceModels \
    -lturbulenceModels \
```

Recompile `myRadiation` with the new multistep soot model

```
wclean
wmake
```

Chapter 4

Case tutorials

In this tutorial, the aachenBomb case is used as the basic case. We will go deep into the case and explain the injection configuration and how to get the parameters for the injection from ECN which is organized by Sandia Lab in US. Also, the chemkin file will be explained. More importantly, the radiationProperties will be modified to provide the soot model constants. Finally, we will crate a multistepSootBomb case to run the sprayFoam solver.

4.1 case modification

For the case introduction, report [7] has already given a detailed description. In this tutorial, we more focus on how to set a case according the parameters from ECN.
Let's copy the aachenBomb case from the tutorials to user directory and rename

```
run
cp -r $FOAM_TUTORIALS/lagrangian/sprayFoam/aachenBomb .
mv aachenBomb multistepSootBomb
cd multistepSootBomb
```

In ECN, the geometry is a cube with length 108 mm, hence we have to modify the `blockMeshDict`. Only the `vertices` and `blocks` need to be changed, the modified part is shown as bellow by typing `vi system/blockMeshDict`

```
vertices
(
    (-54 0 -54)
    (54 0 -54)
    (54 108 -54)
    (-54 108 -54)
    (-54 0 54)
    (54 0 54)
    (54 108 54)
    (-54 108 54)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (54 54 54) simpleGrading (1 1 1)

);
```

As the flame region is far smaller than cube domain, it is better to use the refined mesh for the liquid and vapour region.
Copy the file `Allrun` from the tutorial,

```
cp $FOAM_TUTORIALS/incompressible/simpleFoam/airFoil2D/Allrun .
```

Comment `runApplication` inside of `Allrun`,

```
sed -i s/runApplication/#runApplication/g Allrun
```

To generate the `blockMesh`, we need to insert `blockMesh` command into the `Allrun`,

```
echo 'blockMesh > log.blockMesh 2>&1' >> Allrun
```

To define the fine mesh region, we need to insert `topoSet` command into the `Allrun`,

```
echo 'topoSet > log.topoSet.1 2>&1' >> Allrun
```

To get the fine mesh, we need to insert `refineMesh` command into the `Allrun`,

```
echo 'refineMesh  -overwrite > log.refineMesh.1 2>&1' >> Allrun
```

TopoSet and refine again to get the final refined mesh,

```
echo 'topoSet > log.topoSet.1 2>&1' >> Allrun
echo 'refineMesh  -overwrite  > log.refineMesh.1 2>&1' >> Allrun
```

As there is lacking of `topoSetDict` in aachenBomb case, we need to copy it from other case in tutorial and change it,

```
cp $FOAM_TUTORIALS/combustion/fireFoam/LES/smallPoolFire2D/system/topoSetDict system
```

Change the name of topoSet region from `f0` to `refineCells`,

```
sed -i s/f0/refineCells/g system/topoSetDict
```

Change the `faceSet` to `cellSet`,

```
sed -i s/faceSet/cellSet/g system/topoSetDict
```

Change the `boxToFace` to `boxToCell`,

```
sed -i s/boxToFace/boxToCell/g system/topoSetDict
```

Set the reasonable refined region for the current case,

```
sed -i s/'-0.0529 -0.001 -0.1'/'-0.008 0.03 -0.008'/g system/topoSetDict
sed -i s/'0.0529 0.002 0.1'/'0.008 0.108 0.008'/g system/topoSetDict
```

Copy the `refineMeshDict` from other case,

```
cp $FOAM_TUTORIALS/combustion/fireFoam/LES/oppositeBurningPanels/
system/refineMeshDict system
```

Spray properties is one of the most important parts in spray combustion. All the spray parameters are specified in file contant/sprayCloudProperties. We can search the relevant data from [1].Figure 4.1 give a example for ECN data base. You can choose different fuel, density, temperature and so on.

According to the parameters provided in ECN, we give a part of modified codes:injectionModels which is a model setting inside of sprayCloudProperties file." The massTotal has been changed from 6e-6 to 18e-6. outerDiameter has been changed from 1.9e-4 to 10e-5. Duration has been changed from 1.25e-3 to 0.0068. Cd is changed from 0.9 to 0.8.

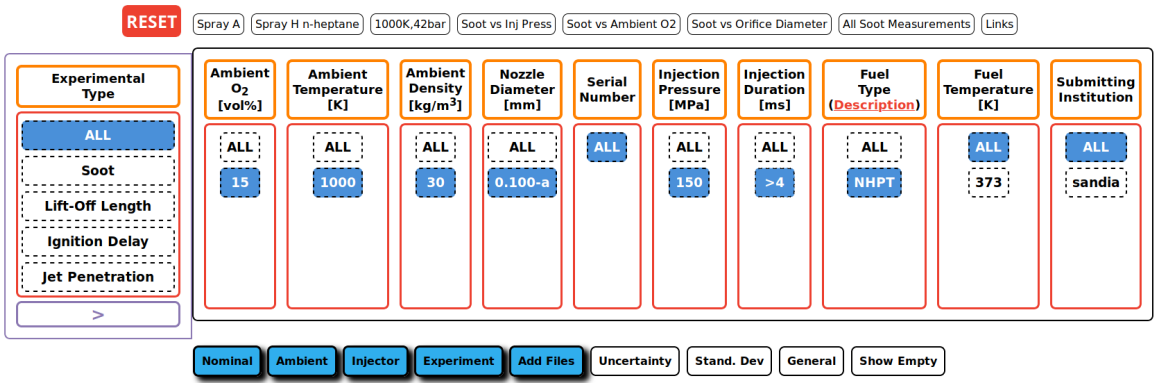

Figure 4.1: The example of data base in ECN

It should be note that the `flowType` is changed to `pressureDrivenVelocity`.Inject pressure: `Pinj 1557e5+` has been added after `flowtype pressureDrivenVelocity`.
Basically, for the `RosinRammlerDistribution`, the minValue equals to the diameter of injector, the maxValue is larger one order than the diameter of injector, the meanValue is around the 70 percent of maxValue. Meanwhile, we also need to change `dispersionModel` from `none` to `stochasticDispersionRAS` and `stochasticDispersionRAS` to `none`. `T0` is changed to 373. The modified injectionModels are given as following,

```
...
injectionModels
 {
     model1
     {
       type            coneNozzleInjection;
         SOI             0;
         massTotal       18e-06;
         parcelBasisType mass;
         injectionMethod disc;
         //flowType        flowRateAndDischarge;
         flowType        pressureDrivenVelocity;
         Pinj     1557e+5;//
         outerDiameter   10e-5;
         innerDiameter   0;
         duration        0.0068;
         position        ( 0 0 0 );
         direction       ( 0 -1 0 );
         parcelsPerSecond 16000000; //baseline:20000000
         flowRateProfile table
         (
            (0           0)
            (0.00008     1)
            (0.00672     1)
            (0.00680     0)
         );

         Cd              constant 0.8;
```

```
            thetaInner      constant 0.0;
            thetaOuter      constant 10.0;

            sizeDistribution
            {
                type        RosinRammler;

                RosinRammlerDistribution
                {
                    minValue        10e-6;
                    maxValue        10e-5;
                    d               7e-5;
                    n               3;
                }
            }
        }
    }
    ...
```

The radiationProperties under the constant folder contains the inputs of radiation model and other submodels. The submodels include absorptionEmissionModel, scatterModel, transmissivityModel, sootModel. To evoke the soot model, the radiation model has to be switched on. The constants for multistepSoot model also need to be provided. The modified file is given as follow,

```
radiation       on;
radiationModel  P1;
solverFreq      10;
absorptionEmissionModel none;
scatterModel    none;
transmissivityModel  none;
sootModel multistepSoot<gasHThermoPhysics>;
multistepSootCoeffs
{
Sct 0.7;
MWc 12;
wPrec 26;
rhoS 2000;
Ca 10000;
Ta 21000;
Cb 3;
Cg 21;
Csg 6000;
Tg 12100;
m 0.5;
q 1.4;
Cw1 127;
CollEff 0.13;
Cw2 10000;
Tw2 19800;
NA  6.02214e+26;
}
```

The radiation model needs to a input file G under the 0 folder and a boundaryRadiationProperties file under the constant folder. This is done as follows,

```
cp $FOAM_TUTORIALS/combustion/fireFoam/LES/smallPoolFire3D/0/G 0
```

```
cp $FOAM_TUTORIALS/combustion/fireFoam/LES/smallPoolFire3D/constant/
   boundaryRadiationProperties constant
```

Note that the new multistepSoot model have two unknown variables(soot and PDSoot) which must be read from the 0 folder. Hence, we should copy or create these two file. This is done as following,

```
cp 0/O2 0/soot
cp 0/O2 0/PDSoot
sed -i s/O2/soot/g 0/soot
sed -i s/O2/PDSoot/g 0/PDSoot
sed -i s/'0 0 0 0 0 0 0 '/'-1 0 0 0 1 0 0 '/g 0/PDSoot
```

To solve the two transport equations, we have to add the discredited scheme and iteration solver for soot and PDSoot in the system/fvScheme and system/fvSolution. In the system/fvScheme, we add following codes after the div(phi,Yi_h) Gauss upwind;,

```
div(phi,soot)   Gauss upwind;
div(phi,PDSoot) Gauss upwind;
```

In the system/fvSolution, the following codes are included,

```
    "(soot|PDSoot)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0;
    }

    "(soot|PDSoot)Final"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0;
    }
```

The entry Gfinal should also be included in system/fvSolution file. It should be include in the solvers after the hFinal,

```
    GFinal
    {
        solver PCG;
        preconditioner DIC;
        tolerance 1e-05;
        relTol 0.1;
    }
```

In the aachenBomb case, only one step chemical reaction is offered, which is extremely not accurate to predict the combustion properties and soot production. In this tutorial, we will provide a way to include a couple reactions under the chemkin folder. Here, we use the Liu mechanism with 44 species and 112 reactions. You can download this mechanisms from [2]. Also, you can find other mechanism form this address. The chem.inp and therm.dat should be replaced by the download file (Also, you can use the chemkin folder provided by this tutorial). There are three different contents inside the chem.inp including the ELEMENTS, SPECIES and REACTIONS. For the meaning of number in REACTIONS, you can find it from Report [5]. Similarly, the tutorial for therm.dat and transportProperties also can be found in Report [5]. For the runTime control, open the system/controlDict and set deltaT as 2e-7, set adjustTimeStep to no.

4.2 case simulation and results

Before the running case, we have to dynamically link the `libmyradiationModels` library so that it can be called by the `sprayFoam` solver. To complete this link, we open the file `system/controlDict` by following commands,

```
vi system/controlDict
```

And then add the following as the last line code in `system/controlDict`,

```
libs ("libmyradiationModels.so");
```

As the time step and grid size are quite small to capture the chemistry phenomena, using the parallel calculation is recommended to reduce the computational time. Thus, we can use the `decomposeParDict`.

```
cp $FOAM_TUTORIALS/compressible/rhoPimpleFoam/RAS/squareBendLiq/
system/decomposeParDict system
```

As the mesh structure is simple, simple decomposition has been selected

```
sed -i s/'hierarchical;'/'simple;'/g system/decomposeParDict

sed -i s/'8 1 1'/'1 8 1'/g system/decomposeParDict
```

This command is used to change the decomposition direction Finally, we need to generate the mesh and run the case by using the following codes,

```
./Allrun
decomposePar
mpirun -np 8 sprayFoam -parallel
```

Figure 4.2 shows the temperature distribution calculated by the Liu 44 mechanism. The highest temperature is around $2300K$.

Figure 4.3 shows the distribution of $C2H2$ concentration, the $C2H2$ is the soot precursor which determine the soot mass and distribution directly. The maximum $C2H2$ concentration is around 1.9e-2. The shape of $C2H2$ has a big head and two tails.

From figure 4.4, we can see the soot distribution is similar to the $C2H2$ distribution. The maximum soot concentration is one order smaller than the $C2H2$. But the shape of soot is smaller than $C2H2$, this is because the oxidation by OH and $O2$.

Figure 4.5 shows the distribution of OH, which encircles the soot. OH is a product of high-temperature chemistry. The region of OH corresponds to the high temperature region in flame. You can see the shape of red region in Figure 4.2 is almost the same as the shape of OH.

Figure 4.6 shows the distribution of $O2$ where the $O2$ is consumed in the region of combustion. Thus, you can only see the very low value in the combustion region.

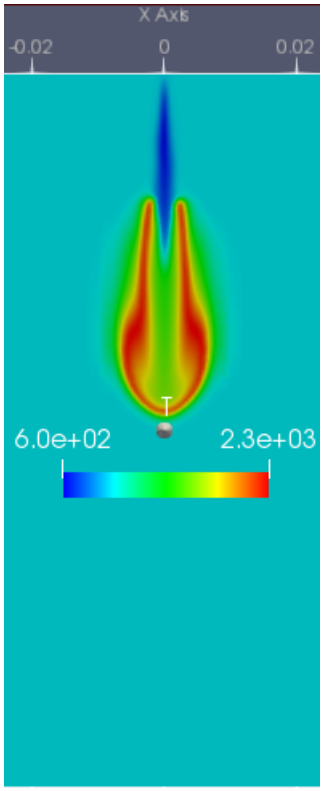

Figure 4.2: Temperature field at 1.4 ms

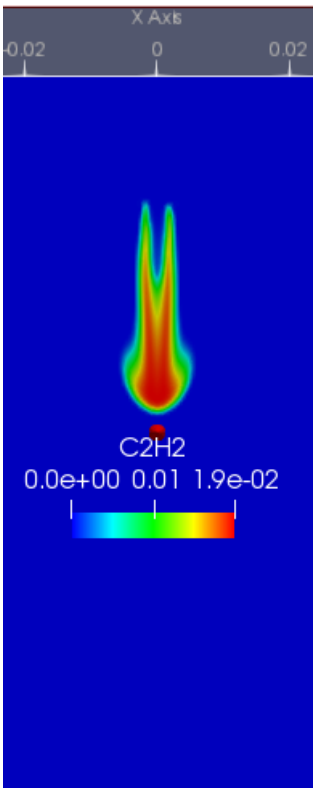

Figure 4.3: C2H2 field at 1.4 ms

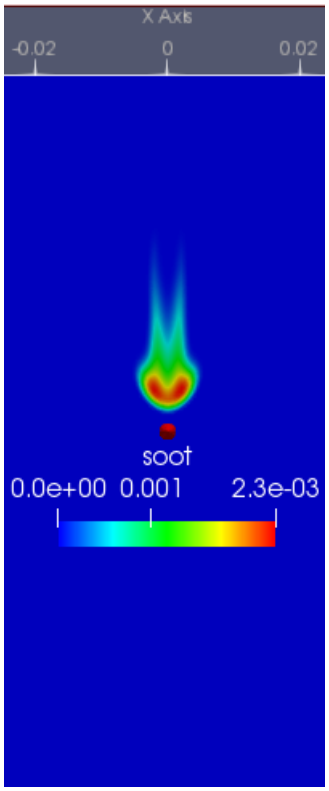

Figure 4.4: soot field at 1.4 ms

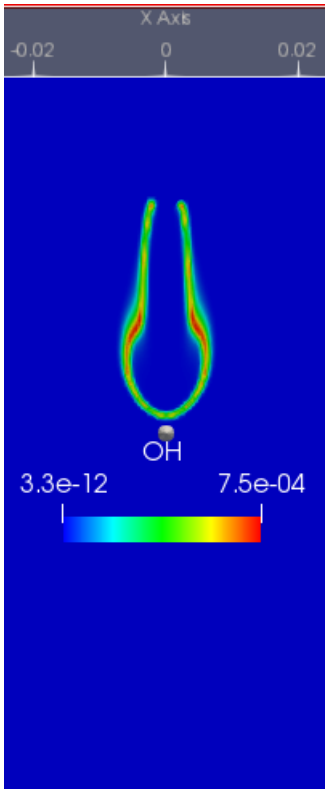

Figure 4.5: OH field at 1.4 ms

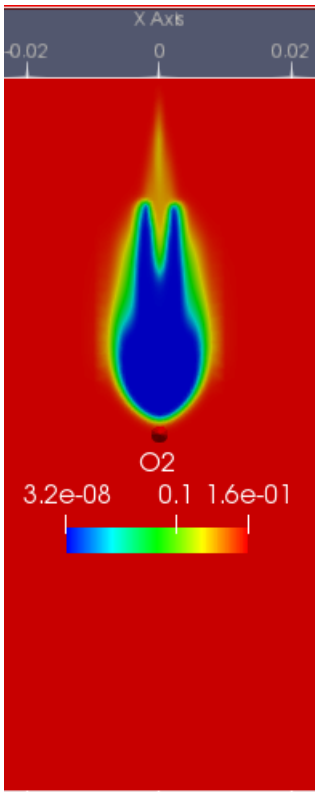

Figure 4.6: O2 field at 1.4 ms

Study questions

1. How to set injector parameters according the data from ECN?
2. How to solve the two transport equations in this tutorial?
3. How to invoke the multistepSoot model?
4. How and where can the multistepSoot model read the submodel constant?
5. how to read the effective turbulent viscosity from turbulence model?
6. how to read initial value from the 0 folder in a case?

Bibliography

[1] URL: `https://ecn.sandia.gov/ecn-data-search`.

[2] URL: `+https://web.stanford.edu/group/+`.

[3] Jiun Cai Ong. "Development of Lagrangian soot tracking method for the study of soot morphology in diesel spray combustion". PhD thesis. University of Nottingham, 2017.

[4] Kar Mun Pang et al. "Evaluation and optimisation of phenomenological multi-step soot model for spray combustion under diesel engine-like operating conditions". In: *Combustion Theory and Modelling* 19.3 (2015), pp. 279–308.

[5] Carlsson Per. "A dieselFoam tutorial". In: *In Proceedings of CFD with OpenSource Software, 2008* (2008).

[6] Pandian Vignesh. "Implementation of soot model for aachenBomb tutorial". In: *In Proceedings of CFD with OpenSource Software* (2016).

[7] S.J Xu. "he implementation of a stochastic reactor (StoR) combustion mode". In: *In Proceedings of CFD with OpenSource Software* (2018).