An FGMFoam tutorial

Michael Bertsch

Gothenburg, Sweden

2019-11-27

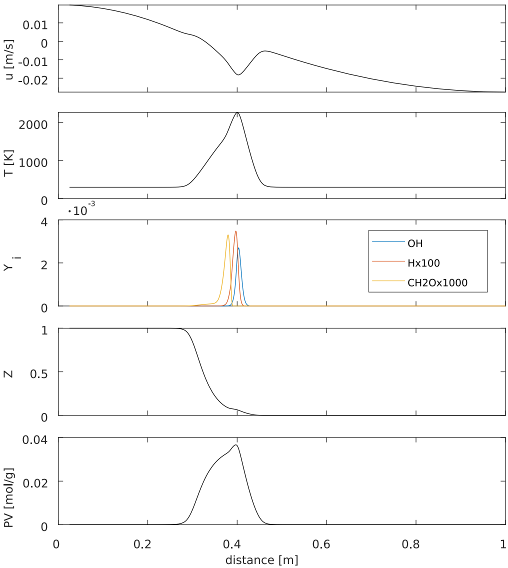
Prerequisites

To follow the tutorial

- FGMFoam solver for OpenFOAM v7 (provided)
  developed by Likun Ma for v2.3.1
- 4D FGM tables (provided)
- SandiaD_LTS tutorial case from OpenFOAM v7

To modify the look-up tables for custom application (optional)

- chemical kinetics software to generate flamelet data, e.g. Cantera
- script to generate 4D tables from flamelet data: contact Likun Ma



flamelet:

1D counter-flow CH4/air flame

The theory of it

The mixture fraction can be defined as

$$Z = \frac{b - b_o}{b_f - b_o}, \tag{1}$$

where the subscripts f and o denote fuel and oxidiser, and b is the Bilger coefficient

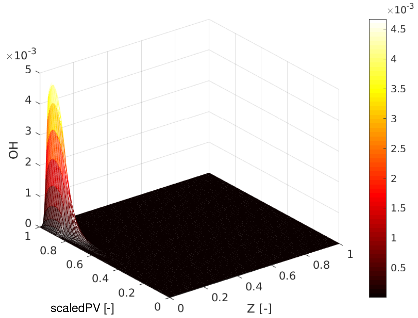
$$b = 2\frac{Y_C}{M_C} + \frac{1}{2}\frac{Y_H}{M_H} - \frac{Y_O}{M_O}. \tag{2}$$

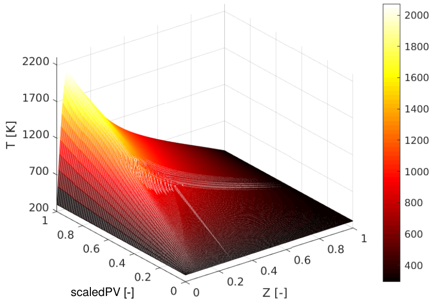
The progress variable, PV, is selected by the author [1] to be

$$PV = 4\frac{Y_{H_2O}}{M_{H_2O}} + 2\frac{Y_{CO_2}}{M_{CO_2}} + \frac{1}{2}\frac{Y_{H_2}}{M_{H_2}} + \frac{Y_{CO}}{M_{CO}}. \tag{3}$$

From flamelets to 4D tables

Below are 2D manifolds used for FGSFoam, similar to FGMFoam but with 2D tabulation (Z,PV) instead of 4D (Z,varZ,PV,varPV). FGSFoam solver, tables, and test case are also provided.





p. 132 in Ma 2016

FGMFoam solution process. See Fig. 6.2 in Ma 2016 [1].

FGMFoam.C

```
...
#include "rhoEqn.H"          //calculate density
while (pimple.loop())
{
    #include "UEqn.H"          //calculate the U/p field
    #include "ZEqn.H"          //transport equation for mixture fraction
    #include "PVEqn.H"         //transport equation for progress variable
    while (pimple.correct())
    {
        #include "pEqn.H"      //PISO loop to calculate p
    }
    turbulence->correct();     //turbulence modelling
}
...
```

ZEqn.H

```
fvScalarMatrix ZEqn
(
    (
      fvm::ddt(rho, Z)
    + fvm::div(phi, Z)
    - fvm::laplacian( (turbulence->muEff()), Z)
    )
);
fvScalarMatrix& ZEqn = tZEqn.ref();
ZEqn.relax();
ZEqn.solve("Z");

turbulence->correctChiZ();
turbulence->correctVarZ();
reaction->correct();
```

PVEqn.H

```
fvScalarMatrix PVEqn
(
    (
      fvm::ddt(rho, PV)
    + fvm::div(phi, PV)
    - fvm::laplacian( (turbulence->muEff()), PV)
    ==
    reaction->SourcePV()
        + fvOptions(rho, PV)
    )
);

fvScalarMatrix& PVEqn = tPVEqn.ref();
PVEqn.relax();
fvOptions.constrain(PVEqn);

PVEqn.solve("PV");
fvOptions.correct(PV);

turbulence->correctChiPV();
turbulence->correctVarPV();
reaction->correct();   //correct sourcePV, T
```

```
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
template<class ReactionThermo>
FGMModel<ReactionThermo>::FGMModel
(
    const word& modelType, ReactionThermo& thermo,
    const compressibleTurbulenceModel& turb,
    const word& combustionProperties
)
:
    sourcePV_(const_cast<volScalarField&>(this->turbulence().sourcePV())),
// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
template<class ReactionThermo>
hashedWordList FGMModel<ReactionThermo>::tables()
{
    tableNames.append("SourcePV"); // Read source term of PV
}
template<class ReactionThermo>
void FGMModel<ReactionThermo>::correct()
{
    scalarField& sourcePVCells = sourcePV_.ref();
    forAll(ZCells, cellI) //- Update i.a. sourcePV internal field
        {
        sourcePVCells[cellI] = solver_.interpolate(ubIF_, posIF_, 4);
        }
    forAll(T_.boundaryField(), patchi)    // Interpolate for patches
        {
         fvPatchScalarField& psourcePV = sourcePV_.boundaryFieldRef()[patchi];
         forAll(pZ , facei)
             {
             psourcePV[facei] = solver_.interpolate(ubP_, posP_, 4);
             }
        }
}
```

$LIB_FGM_SRC/combustionModels/FGMModel/FGMModel.C

SandiaD_LTS tutorial case

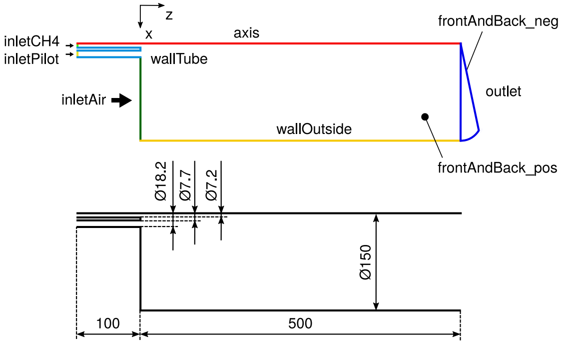


Figure: Schematic representation of the numeric domain in the SandiaD_LTS tutorial case. Dimensions in mm.

Getting started

- Copy the tutorial from OpenFOAM v7.

```
cp -r $FOAM_TUTORIALS/combustion/reactingFoam/RAS/SandiaD_LTS $F
mv $FOAM_RUN/SandiaD_LTS SandiaD_LTS_FGM
```

- Copy the provided FGM tables to

```
$FOAM_RUN/02_table
```

It contains 22 tables in OpenFOAM format, thereof 7 species (47 species reduced mechanism for table generation, but only tables for 7 are provided here). The rest are tabulated thermal, combustion, and fluid properties.

Chemistry: constant/combustionProperties

Change from EDC combustion model to

```
combustionModel  FGMModel;

active  true;

FGMModelCoeffs
{
        useProgressVariableVariance true;
}
```

Chemistry: constant/thermophysicalProperties

Replace the last section of `constant/thermophysicalProperties` with

```
chemistryReader chemkinReader;

CHEMKINFile "$FOAM_CASE/chemkin/reactions.inp";

CHEMKINThermoFile "$FOAM_CASE/chemkin/therm.dat";

CHEMKINTransportFile "$FOAM_CASE/chemkin/transportProperties";
```

Then copy these three files from the FGMFoam example case (provided) to the indicated paths.

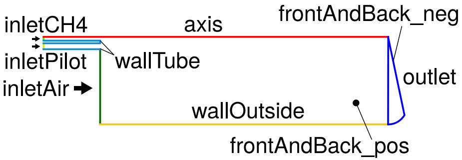
```
mixtureFractionDefinition    "readFromTable";
interpolationType            linearInterpolation;
operatingPressure            1e5;
tablePath                    "$FOAM_RUN/02_table/";

varPV_param
21
(
0
...
1
)
PV_param
51
(
0
...
1
)
varZ_param
21
(
0
...
1
)
Z_param
51
(
0
...
1
);
```

constant/tableProperties

Chemistry: `constant/PVtableProperties`

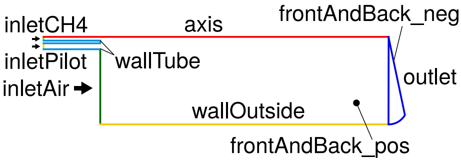
```
mixtureFractionDefinition    "readFromTable";
interpolationType            PVlinearInterpolation;
operatingPressure            1e5;
tablePath                    "$FOAM_RUN/02_table/";

varZ_param
21
(
0
...
1
)
Z_param
51
(
0
...
1
);
```

Boundary and initial conditions



- rename the field `0/T.orig` to *T*
- copy the *CH4.orig* file and rename it to *Z*
- The mixture fraction is 0 at inletAir, 0.04293 at inletPilot, and 0.1559 at inletCH4.
- Create another copy, rename it *varZ* and set all *fixedValue* entries to 0.
- Repeat this procedure for *PV*, *varPV* and *sourcePV*.
- sourcePV, has the unit $kg\, m^{-3}\, s^{-1}$, therefore set the dimensions to [1 -3 -1 0 0 0 0].
- Repeat the procedure once more for *scaledPV*, dimensionless, but in this case set the value of scaledPV at inletPilot to 1.

Boundary and initial conditions:



	d_i	D	U_Z	turb. intensity	mix. length	T	ϕ
inletCH4	-	7.2	49.6	0.0458	5.04e-4	294	3.17
inletPilot	7.7	18.2	11.4	0.0628	7.35e-4	1880	0.77
inletAir	18.2	300	0.9	0.0471	0.019677	291	0

Table: Inner and outer diameter in mm, axial velocity component in m/s, turbulence intensity, mixing length in m, temperature in K, and equivalence ratio at the three inlet boundaries.

Clean the 0 directory by removing all species mass fractions, Ydefault, G, and nut. The final test case is provided for comparison.

system/fvSchemes

- Set the time integration scheme to Euler.
- Add these lines under *divSchemes*

```
div(phi,Z)        Gauss upwind;
div(phi,varZ)     Gauss upwind;
div(phi,PV)       Gauss upwind;
div(phi,varPV)    Gauss upwind;
div(phi,K)        Gauss upwind;
div(phi,Yi_h)     Gauss limitedLinear 1;
div(phi,nuTilda) Gauss limitedLinear 1;
div((muEff*dev2(T(grad(U))))) Gauss linear;
div(phid,p)       Gauss limitedLinear 1;
```

system/fvSolution

```
solvers
{
    "rho.*"
    {
        solver          diagonal;
    }
p PCG
    {
        preconditioner GAMG
        {
            tolerance        2e-5;
            relTol           0.05;
            nVcycles         2;
            smoother         GaussSeidel;
            nPreSweeps       0;
            nPostSweeps      2;
            nFinestSweeps    2;
            cacheAgglomeration false;
            nCellsInCoarsestLevel 300;
            agglomerator     faceAreaPair;
            mergeLevels      1;
        };

        tolerance        1e-6;
        relTol           0.05;
        maxIter          100;
    };
pFinal PCG
    {
        preconditioner GAMG
        {
            tolerance        2e-5;
            relTol           0.05;
            nVcycles         2;
            smoother         GaussSeidel;
            nPreSweeps       0;
            nPostSweeps      2;
            nFinestSweeps    2;
            cacheAgglomeration false;
            nCellsInCoarsestLevel 300;
            agglomerator     faceAreaPair;
            mergeLevels      1;
        };

        tolerance        1e-6;
        relTol           0;
        maxIter          100;
    };
    "(U|h|Z|PV|k|epsilon)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0.01;
    }
    "(U|h|Z|PV|k|epsilon|varZ|varPV)Final"
    {
        $U;
        relTol          0;
    }
```

system/fvSolution

```
...

PIMPLE
{
    momentumPredictor yes;
    nOuterCorrectors  1;
    nCorrectors      2;
    nNonOrthogonalCorrectors 1;
}
```

system/controlDict

- Set
  - `deltaT         5e-5;`
  - `writeInterval  0.1;`
  - `endTime        0.1;`
- run `blockMesh`
- run the FGM solver by typing `FGMFoam`
- An Allrun script is provided to run and postprocess the case.

Postprocessing

- Create a copy of the last time directory.
- Now run the postprocessing application

```
FGMFoamPost -latestTime
```

- This gives all species specified in `$FOAM_CASE/chemkin/reactions.inp`. With the provided tables, only the following species can be looked up in the tables:

```
SPECIES
H2O CO2 O2 CH4 OH CH2O N2
END
```

Results

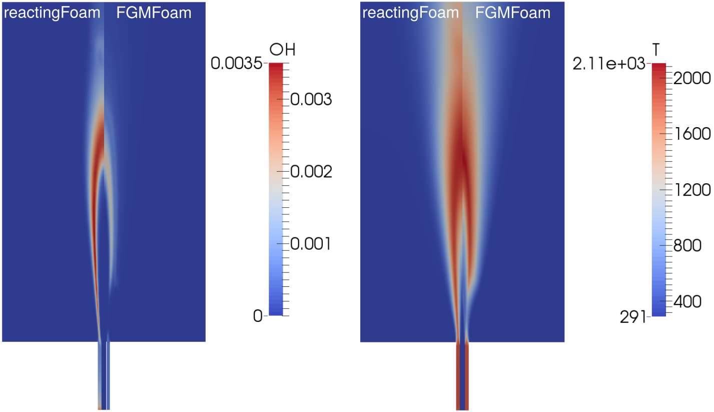


Figure: Comparison of results for the SandiaD_LTS tutorial case from reactingFoam and FGMFoam.

References

[1] L. Ma, *Computational modelling of turbulent spray combustion*.
PhD thesis, Delft University of Technology, 2016.
https://doi.org/10.4233/uuid:c1c27066-a205-45f4-a7b4-e36016bc313a.