Cite as: Bertsch, M.: Description of the reacting flow solver FGMFoam. 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

Description of the reacting flow solver FGMFoam

Developed for OpenFOAM v7
Requires: 4D FGM tables,
FGMFoam solver (both provided)

Author:
Michael Bertsch
Lund University

Peer reviewed by:
Shannon Leakey

January 30, 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 FGMFoam reacting flow solver implemented by Likun Ma [1].
- how to generate the necessary flamelet data for the test case (optional).

The theory of it:

- briefly about the implementation of the reactingFoam solver, by means of an example case.
- about the general theory of the flamelet modelling approach.
- about possible ways to generate 1D flamelet data.
- about the choice of mixture fraction and progress variable.
- about the limitations of this approach compared to other reacting flow solvers/combustion chemistry reduction techniques.

How it is implemented:

- about the principles of the flamelet-based solver, and about the added transport equations for mixture fraction and progress variable, as well as the source term. It is described how selected fluid parameters are updated based on flamelet look-up tables.

How to modify it:

- how to modify the reactingFoam solver in OpenFOAM v7 to a solver based on the flamelet combustion model in a step-by-step tutorial. The effects of the modifications will be tested by comparison to the results from the original reactingFoam solver, for the SandiaD_LTS tutorial case.

Prerequisites

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

- Run standard document tutorials like the SandiaD_LTS reactingFoam tutorial.
- Chapter 6 of Likun Ma's PhD thesis [1]: Implementation and validation of LES/FGM method in OpenFOAM.

Contents

Chapter 1

Introduction

Solving the transport equations for a large number of species can be computationally expensive when investigating detailed chemistry effects in turbulent reacting flows. Instead of solving the species transport equations for a reduced set of species and/or reactions, a tabulated chemistry model can be used to resolve detailed chemistry at lower computational cost. An example for a tabulated chemistry model is the flamelet approach. It is assumed that the properties of the reacting fluid(s) can be accurately described by only a set of variables. This assumption is valid in a certain combustion regime, the laminar flamelet regime. Simulations of 1D flames, so-called flamelets, are used to create tables for selected fluid and flow properties as a function of the paramaters chosen as tabulation parameters. Flamelet-generated manifolds (FGM) are constructed from a range of 1D combustion simulations. Part of the set of tabulation parameters are often a mixture fraction and a progress variable. In the following it is described how to modify the reactingFoam solver in OpenFOAM v7 to such an FGM tabulated chemistry reacting flow solver, called FGMFoam. This solver has been implemented by Likun Ma in OpenFOAM v2.3.1 [1]. The implementation is described in the PhD thesis of Ma 2016 [1]. After these step-by-step instructions how to add FGMFoam to the collection of reacting flow solvers in OpenFOAM v7, a tutorial case describes the usage of this solver. The performance of FGMFoam is compared to that of reactingFoam based on the SandiaD_LTS tutorial case.

Chapter 2

Description of reactingFoam

2.1 The reactingFoam solver

A detailed description of the reacting flow solver reactingFoam is given, for example, in Andreas Lundström's tutorial [2] or in Andersen 2008 [3]. Transport equations for N-1 species are solved in addition to the conservation equations for mass, momentum, and energy. Even for reduced reaction mechanisms, solving these additional equations for every time step can be a limiting factor for the numeric investigation of reacting flows. The simplified structure of the reactingFoam solver is as follows

```
...
#include "rhoEqn.H"              //calculate density
while (pimple.loop())
{
    #include "UEqn.H"            //calculate the U/p field
    #include "YEqn.H"            //species transport equation with reaction source term
    #include "EEqn.H"            //determine T from chemical reaction enthalpy lookup
    while (pimple.correct())
    {
        #include "pEqn.H"        //PISO loop to calculate p
    }
    turbulence->correct();       //turbulence modelling
}
rho = thermo.rho();              //update rho from T
...
```

The species transport equation contains the source term 'reaction->R(Yi)'. $R(Yi)$ is the reaction source term calculated in the combustion model. The combustion model refers to the chemistry model which ultimately provides the source term for the species transport equation [4].

2.2 The SandiaD_LTS tutorial case

The SandiaD_LTS tutorial case is based on a well-documented research burner featuring a piloted diffusion flame [5]. In the following, this tutorial case will be explained for reference when describing the adaptation of this tutorial case for the flamelet solver FGMFoam in Section 4.1.2. As a first step, copy the SandiaD_LTS tutorial case to the run directory.

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

Apart from an Allrun script, SandiaD_LTS contains the directories 0, constant, system and chemkin.

Initial and boundary conditions (0)

The 0 directory contains pressure p, velocity U, turbulence kinetic energy k, turbulence dissipation rate ϵ, turbulence viscosity ν_t, and turbulent thermal diffusivity α_t. Also, temperature T and incident radiation G, as well as the mass fractions of CO, CO_2, H, H_2, H_2O, N_2, O, O_2 and OH. The initial and boundary conditions for the mass fractions of all other included species are set by Ydefault.

The computational domain is a wedge representing the axisymmetric setup. The boundaries and dimensions of the domain are shown in Fig. 2.1. A rich methane-air mixture with 25 vol-% CH_4 enters the domain at inletCH4, corresponding to a fuel-air equivalence ratio ϕ of 3.17. The diameter of inletCH4 is 7.2 mm. Both inletCH4 and inletPilot are modelled from 100 mm upstream of the expansion to wallOutside. An overview over selected inlet boundary conditions is given in Table 2.1.

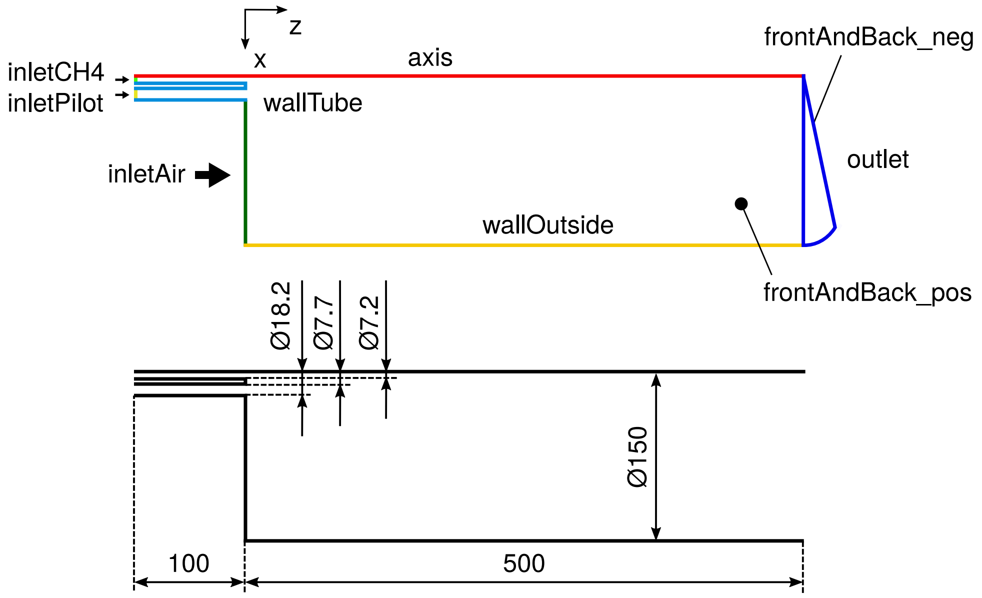

Figure 2.1: Schematic representation of the numeric domain in the SandiaD_LTS tutorial case. Dimensions in mm.

Table 2.1: 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.

	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

The lean combustion process of the pilot flame is not solved, instead an inlet condition corresponding the product of the combustion process is prescribed at inletPilot. The inlet temperature at the pilot inlet is set to 1880 K, and the product concentration was calculated separately for a premixed unstrained flame at ϕ=0.77, with the mass fractions of the major species and radicals given in table 2.2. At inletAir, air at a temperature of 291 K enters the domain at a speed of 0.9 $\frac{m}{s}$.

Table 2.2: Species mass fractions at the three inlet boundaries.

	N_2	O_2	CH_4	CO_2	H_2O	H_2	OH	CO	O	H
inletCH4	0.6473	0.1966	0.1561	0	0	0	0	0	0	0
inletPilot	0.7342	0.054	0	0.1098	0.0942	1.29e-4	2.8e-3	4.07e-3	7.47e-4	2.48e-5
inletAir	0.77	0.23	0	0	0	0	0	0	0	0

A no-slip condition is specified at wallTube, while zero velocity gradient is specified at wallOutside. Wall functions are applied for both wall patches for k, ϵ and ν_t. The outlet is modelled as a 1e5 Pa total pressure outlet.

Model properties (constant)

- chemistryProperties.orig: Under chemistryType, the solver ode is selected to solve ordinary differential equations. Tabulation of dynamic adaptive chemistry (TDAC) is chosen as solution method to speed up chemistry calculations. This feature was added in OpenFOAM-v5. It was demonstrated that this chemistry reduction can increase the efficiency of solving species transport in case of modestly detailed reaction mechanisms like GRI-Mech. 3.0 [6]. The initial chemical time step size is set to 1e-7 s, which is independent of the flow time step size specified in system/controlDict. Under odeCoeffs, the ODE solver 'seulex' is selected, and absolute and relative tolerances are specified. Under importantSpecies, the species which are to be left together with elementary reactions at the end of the reduction process are specified [7]. Finally, the tolerances for reduction and tabulation are specified.

- combustionProperties: In this file, the eddy dissipation concept (EDC) turbulent combustion model is selected. Under EDCCoeffs, the version of the EDC model is specified, in this tutorial 'v2005'. For this particular version, C$_\gamma$=2.1377 and C$_\tau$=0.4083 in

$$\overline{R}_i = \frac{\overline{\rho}}{\tau^*}\frac{\gamma_L^2\ \chi}{1-\gamma_L^2\ \chi}\left(\overline{Y_i} - Y_i^*\right). \tag{2.1}$$

  Here, $\overline{R}_i$ is the mean reaction rate of species i, $\overline{\rho}$ is the mean density, χ is the reaction fraction of the fine structures, $\overline{Y_i}$ is the mean mass fraction of species i, and Y_i^* is the fine structure mass fraction of species i. Further,

$$\gamma_L = C_\gamma Re_t^{-1/4} \tag{2.2}$$

  is the structure length fraction, and

$$\tau^* = C_\tau Re_t^{-1/2}\frac{k}{\epsilon} \tag{2.3}$$

  denotes the fine structure residence time, based on the turbulent Reynolds number Re_t [8].

- fvOptions: In this file it is specified that the radiation library 'libradiationModels.so' will be loaded. The model settings for the radiation model are then set in *radiationProperties*.

- radiationProperties: This file includes a switch to decide whether radiation is considered in the simulation. In the current case, the P1 radiation model is selected. The frequency at which radiation iterations are to be carried out is specified relative to the number of flow iterations. In this tutorial case it is selected to have one radiation iteration for every single flow iteration. A radiation absorption and emission submodel is specified, in this case greyMeanCombustion. This model gives the radiation and emission coefficients for continuous phase. Model coefficients need to be supplied for all species that are being solved, i.e. which are not included in a lookup-table for absorption and emission coefficients. In this tutorial case, no such lookup-table is used, instead, the model parameters are supplied in the form of polynomial coefficients for two fifth order polynomials for a low and a high temperature range. The temperature limit between low and high temperature region is also specified. In this tutorial case, only the limit temperature is set to 200 K, therefore only the coefficients for the high temperature region need to be specified, the low temperature coefficients are dummy 0 values. In this case, neither scatter nor soot is considered in the radiation absorption and emission submodel.

- g: Gravity is taken into account in this tutorial case. The gravitational acceleration is specified in this file.
- reactionsGRI: 5 elements, 36 species, 218 reactions, and the valid temperature range are specified in this file.
- thermo.compressibleGasGRI: Contains information in OpenFOAM format for 53 species, including molar weight, polynomial coefficients for the specific heat capacity, transport coefficients, and elementary composition.
- thermophysicalProperties: Specification of an inert specie, in this case N_2. This species is calculated explicitly as $Y_{inert} = 1 - \sum Y_{active,i}$. The type of chemistry reader, as well as the paths to the above described chemistry (reactionsGRI) and chemistryThermo (thermo.compressibleGasGRI) files are specified. Furthermore, seven options in the thermoType package are set:
    - Thermophysical models: For reactingFoam, the hePsiThermo model class is selected. The reactingFoam solver constructs the model classes psiThermo as well as psiReactionThermo. A fixed composition is assumed, and they are based on the compressibility $\psi = (RT)^{-1}$, where R is the universal gas constant.
    - Mixture model: For the reactingFoam solver, mixtures with variable composition have to be considered, therefore the reactingMixture model is selected. This model requires the model coefficients provided in the chemistry file 'reactionsGRI'.
    - Transport model: One of four transport models is selected. In this case is the Sutherland model, described in the following section.
    - Thermodynamic model: This model is used to determine the specific heat capacity c_p. In this tutorial case, 'janaf' is selected. In this model, c_p is determined based on tabulated coefficients as
    $$c_p = R\left(\left(\left(\left(a_4 T + a_3\right) T + a_2\right) T + a_1\right) T + a_0\right). \tag{2.4}$$
    - Energy variable: In this case, the sensible enthalpy h_s is selected as variable in the energy equation.
    - Equation of state: In the tutorial case, the perfect gas law
    $$\rho = \frac{p}{RT} \tag{2.5}$$
    is incorporated in the model.
    - Composition: The keyword 'specie' refers to the submodel describing the molar weight of each specie.
- turbulenceProperties: The RANS model kEpsilon is selected for turbulence modelling.

A detailed description of the required files in the constant directory for reactingFoam can be found, for example, in Haddadi et al. 2015 [9].

Chemistry and transport properties (chemkin)

In constant/thermophysicalModels 'sutherland' is specified as transport model, therefore the Sutherland coefficient A_s and the Sutherland temperature T_s need to be specified in chemkin/transportProperties for each of the species. The Sutherland model gives the dynamic viscosity as

$$\mu = \frac{A_s\sqrt{T}}{1+\frac{T_s}{T}}, \tag{2.6}$$

where T is the variable temperature. This transport model also gives the thermal conductivity κ and the thermal diffusivity α. A_s=1.512e-6 $Pa\ s\ K^{-0.5}$ and T_s=120 K are specified for all species but H_2 and CO_2, for which the model parameters are set individually. The grimech30.dat file contains the description of the reaction mechanisms with corresponding rate constants, altogether 325 reactions for 53 species. Thermo30.dat contains the corresponding thermochemical data, in form of NASA polynomial coefficients for each specie. The utility *chemkinToFoam* converts the three files, grimech30.dat and thermo30.dat in chemkin-II format, and transportProperties, to reactionsGRI and thermo.compressibleGasGRI in the *constant* directory.

Solution settings (system)

In system/fvSchemes the divergence schemes for transport of species mass fractions Yi and species enthalpy Yi_h are set. The '01' in limitedLinear**01** assures stronger bounding between 0 and 1.

Chapter 3

Implementation of FGMFoam

The FGMFoam solver is based on the flamelet assumption for turbulent combustion. The turbulent flame is modelled locally as a laminar flame. The parameters of adequate laminar flames are determined before the 2D/3D simulation from 1D combustion simulations. The data from these 1D flames, so-called flamelets, are then used in a second stage to generate a table as a function of suitable tabulation variables, depending on the application. Often one tabulation parameter is the so-called progress variable PV, between 0 (unburned state, reactants) and 1 (completely reacted, products). The mixture fraction, in this document denoted by Z, can be used to describe the composition of the air/fuel mixture. In order to improve the model accuracy, additional tabulation parameters can be selected, such as the scalar dissipation rate. In the presented implementation of the FGMFoam solver, PV, Z, and their respective variance are used as control parameters to generate 4D tables from 1D flame parameters. Figure 3.1 shows a flow chart for the solution algorithm of the FGMFoam solver.

p. 132 in Ma 2016

Figure 3.1: Flow chart of the FGMFoam solver solution algorithm. Shown in Ma 2016 [1].

The simplified structure of the FGMFoam solver, from *FGMFoam.C*, is shown in the following.

```
...
#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
}
...
```

The transport equation for the progress variable is implemented in *PVEqn.H* as

```
1   tmp<fvScalarMatrix> tPVEqn
2   (
3       (
4         fvm::ddt(rho, PV)
5       + fvm::div(phi, PV)
6       - fvm::laplacian( (turbulence->muEff()), PV)
7       ==
8       reaction->SourcePV()
9           + fvOptions(rho, PV)
10      )
11  );
12  fvScalarMatrix& PVEqn = tPVEqn.ref();
13  PVEqn.relax();
14  fvOptions.constrain(PVEqn);
15
16  PVEqn.solve("PV");
17  fvOptions.correct(PV);
18  PV = max(   min( PV, 1.0 ), 0.0   ); //bound progress variable
19
20  turbulence->correctChiPV();
21  turbulence->correctVarPV();
22  reaction->correct();   //correct SourcePV, T, sensor
```

The progress variable source term *reaction->SourcePV()* in line 8 is passed from the FGM combustion model as a tabulated value from the look-up table. The transport equation for the mixture fraction is implemented in *ZEqn.H* as

```
Switch IsPremixed(chemistryProperties.lookup("premixed"));
if (!IsPremixed) //do not solve transport equation if perfectly premixed
{
   tmp<fvScalarMatrix> tZEqn
   (
        (
              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();
}
    Z = max(   min( Z, 1.0 ), 0.0   ); //bound mixture fraction
```

3.1 Modification of the reactingFoam solver

In the following, the main steps to generate the directory structure of the FGMFoam solver are shown. This is based on the reactingFoam solver, but relies on additional copy-paste steps from the provided source code of FGMFoam, which is in turn an adaption of the reactingFoam solver implemented by Likun Ma in OpenFOAM v2.3.1. It is described which libraries have to be copied and slightly modified. The final source code for FGMFoam in OpenFOAM v7 is provided for comparison.

3.1.1 Copy the reactingFoam solver

As the first step, copy the reactingFoam solver from the applications directory to the corresponding location in the OpenFOAM user directory, and rename the application FGMFoam.C.

```
export FGM_SOLVER_DIR=$WM_PROJECT_USER_DIR/applications/FGMFoam/applications/solver
mkdir --parents $FGM_SOLVER_DIR
cp -r $FOAM_APP/solvers/combustion/reactingFoam $FGM_SOLVER_DIR
mv $FGM_SOLVER_DIR/reactingFoam $FGM_SOLVER_DIR/FGMFoam
cd $WM_PROJECT_USER_DIR/applications/FGMFoam/applications/solver/FGMFoam
mv reactingFoam.C FGMFoam.C
```

The next step will be to copy and modify all libraries which need to be adapted for the flamelet solver.

3.1.2 Copy and modify libraries

The path to the source file is written to the local *bashrc* file for FGMFoam.

```
echo "export LIB_FGM_SRC=$WM_PROJECT_USER_DIR/applications/FGMFoam/src" > bashrc
source bashrc
```

An *Allwmake* script is prepared to compile all necessary customised libraries and the FGMFoam solver.

```
echo '#!/bin/sh' > Allwmake
echo 'cd ${0%/*} || exit 1    # run from this directory' >> Allwmake
echo 'makeType=${1:-libso}' >> Allwmake
echo 'set -x' >> Allwmake
echo '' >> Allwmake
echo 'wmake $makeType ./thermophysicalModels/reactionThermo' >> Allwmake
echo 'wmake $makeType ./thermophysicalModels/chemistryModel' >> Allwmake
echo 'wmake $makeType ./thermophysicalModels/solidThermo' >> Allwmake
echo 'wmake $makeType ./thermophysicalModels/solidChemistryModel' >> Allwmake
echo '' >> Allwmake
echo 'wmake $makeType ./TurbulenceModels/turbulenceModels' >> Allwmake
echo 'wmake $makeType ./TurbulenceModels/compressible' >> Allwmake
echo '' >> Allwmake
echo 'wmake $makeType ./combustionModels' >> Allwmake
echo 'wmake $makeType ./fvOptions' >> Allwmake
echo '' >> Allwmake
echo 'wmake ../applications/utilities/FGMFoamPost/' >> Allwmake
echo 'wmake ../applications/solver/FGMFoam/' >> Allwmake
echo '' >> Allwmake
echo '# --------------------------------------- end-of-file' >> Allwmake
```

The execute permission is added to the attributes of the *Allwmake* script, and a corresponding *Allclean* script is generated.

```
chmod +x Allwmake
cp Allwmake Allclean
sed -i 's/wmake/wclean/g' Allclean
```

In the following, the changes to each of the libraries previously copied to the OpenFOAM user directory are described step-by-step. FGMFoam is based on modifications of parts of the libraries

- thermophysicalModels,
- TurbulenceModels,
- combustionModels, and
- fvOptions.

Changes to *thermophysicalModels*

As a first step, copy the libraries *chemistryModel*, *reactionThermo*, *solidChemistryModel* and *solidThermo* to the OpenFOAM user directory.

```
mkdir --parents $LIB_FGM_SRC/thermophysicalModels
cp -r
↪  $FOAM_SRC/thermophysicalModels/{chemistryModel,reactionThermo,solidChemistryModel,solidThermo}
↪  $LIB_FGM_SRC/thermophysicalModels
```

Go to the user library directory for FGMFoam and modify *Make/files* for all three thermophysical submodels so that the libraries are created in the user library directory.

```
cd $LIB_FGM_SRC
sed -i 's/FOAM_LIBBIN/FOAM_USER_LIBBIN/g'
↪  thermophysicalModels/reactionThermo/Make/files
↪  thermophysicalModels/chemistryModel/Make/files
↪  thermophysicalModels/solidChemistryModel/Make/files
↪  solidThermo/basic/Make/files
```

Change the name of the created library by adding 'FGMFoam' to the library name for all three submodels to distinguish them from the standard libraries.

```
sed -i '1h;1!H;$!d;x;s/.*lib/&FGMFoam/'
↪  thermophysicalModels/reactionThermo/Make/files
↪  thermophysicalModels/chemistryModel/Make/files
↪  thermophysicalModels/solidChemistryModel/Make/files
↪  thermophysicalModels/solidThermo/Make/files
```

Also update the path variables in *Make/options* for *reactionThermo*, *chemistryModel*, *solidChemistryModel* and *solidThermo*.

```
sed -i 's=LIB_LIBS.*=& \n    -L\$(FOAM_USER_LIBBIN) \\='
↪  thermophysicalModels/reactionThermo/Make/options
↪  thermophysicalModels/chemistryModel/Make/files
↪  thermophysicalModels/solidChemistryModel/Make/files
↪  thermophysicalModels/solidThermo/Make/files
```

Changes to *TurbulenceModels*

Modifications of the *TurbulenceModels* library are also necessary. As a first step, copy the libraries *turbulenceModel* and *compressible* to the OpenFOAM user directory.

```
mkdir --parents $WM_PROJECT_USER_DIR/applications/FGMFoam/src/TurbulenceModels
cp -r $FOAM_SRC/TurbulenceModels/{turbulenceModels,compressible}
↪  $WM_PROJECT_USER_DIR/applications/FGMFoam/src/TurbulenceModels
```

Go to the user library directory for FGMFoam and modify *Make/files* for the two local version of turbulence model source files so that the libraries are created in the user library directory.

```
cd $LIB_FGM_SRC
sed -i 's/FOAM_LIBBIN/FOAM_USER_LIBBIN/g' TurbulenceModels/compressible/Make/files
↪  TurbulenceModels/turbulenceModels/Make/files
```

Again, change the name of the created library by adding 'FGMFoam' to the library name for the two turbulence submodels to distinguish them from the standard libraries.

```
sed -i '1h;1!H;$!d;x;s/.*lib/&FGM/' TurbulenceModels/compressible/Make/files
↪  TurbulenceModels/turbulenceModels/Make/files
```

Also update the path variables in *Make/options* for *turbulenceModel* and *compressible*, which access customised libraries.

```
sed -i 's=LIB_LIBS.*=& \n    -L\$(FOAM_USER_LIBBIN) \\='
↪  TurbulenceModels/compressible/Make/files
↪  TurbulenceModels/turbulenceModels/Make/files
```

The options file in `$LIB_FGM_SRC/TurbulenceModels/turbulenceModels/Make` does not need to be modified. The file `compressible/Make/options`, should then read

```
EXE_INC = \
    -I../turbulenceModels/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_FGM_SRC)/thermophysicalModels/solidThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/solidSpecie/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \

LIB_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lFGMFoamSolidThermo \
    -lsolidSpecie \
    -lFGMFoamTurbulenceModels \
    -lspecie \
    -lfiniteVolume \
    -lmeshTools
```

Changes to *combustionModels*

A modification of the *combustionModel* library is essential to create the FGMFoam application. As a first step, copy the libraries

- combustionModel and
- CombustionModel

to the OpenFOAM user directory. Also copy the *Make* directory.

```
mkdir --parents $WM_PROJECT_USER_DIR/applications/FGMFoam/src/combustionModels
cp -r $FOAM_SRC/combustionModels/{combustionModel,CombustionModel,Make}
↪  $WM_PROJECT_USER_DIR/applications/FGMFoam/src/combustionModels
```

Go to the user library directory for FGMFoam and modify *Make/files* for the two local version of turbulence model source files so that the libraries are created in the user library directory.

```
cd $LIB_FGM_SRC
sed -i 's/FOAM_LIBBIN/FOAM_USER_LIBBIN/g' combustionModels/Make/files
```

Here too, change the name of the created library by adding 'FGMFoam' to the library name for combustion model to distinguish it from the standard combustion library.

```
sed -i '1h;1!H;$!d;x;s/.*lib/&FGMFoam/' combustionModels/Make/files
```

Also update the path variables in *Make/options*, which access customised libraries upstream.

```
sed -i 's=LIB_LIBS.*=& \n    -L\$(FOAM_USER_LIBBIN) \\='
↪  combustionModels/Make/files
```

Copy the directories

- FGMModel
- FGMTable
    - FGMTable
    - linearInterpolation
    - PVlinearInterpolation
    - PVTable
    - PVtableSolver
    - tableSolver

to the *combustionModels* directory. These directories are available for download on the course homepage. They are a modified version of the FGMFoam solver in OpenFOAM v2.3.1 implemented by Likun Ma. Each (sub-)directory contains a source and a header file of same name as the directory. Four dimensional (bilinear) interpolation with the tabulation parameters Z, varZ, PV and varPV is implemented in *linearInterpolation*. The interpolation algorithm is similar to the 3D table interpolation in flameletFoam [10], [11].

As an example for a tabulated parameter, the definition of the source term *SorucePV* in the progress variable equation in `$LIB_FGM_SRC/combustionModels/FGMModel/FGMModel.C` is shown in the following.

```
...
// * * * * * * * * * * * * * * * * 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) // first: for internal field
        {
        ...
        sourcePVCells[cellI] = solver_.interpolate(ubIF_, posIF_, 4);
        }
        ...
    forAll(T_.boundaryField(), patchi)    // then: interpolate for patches
        {
         ...
         fvPatchScalarField& psourcePV = sourcePV_.boundaryFieldRef()[patchi];
         ...
         forAll(pZ , facei)
             {
             ...
             psourcePV[facei] = solver_.interpolate(ubP_, posP_, 4);
             ...
             }
        }
}
template<class ReactionThermo>
Foam::tmp<Foam::volScalarField>
FGMModel<ReactionThermo>::SourcePV() const          //Added L.Ma, 08-10-2014
{
  return sourcePV_;
}
```

Changes to *fvOptions*

Finally, a modified version of the *fvOptions* library is created. As a first step, copy the libraries

- cellSetOption,
- constraints,
- corrections,
- interRegionOption, and
- sources

to the OpenFOAM user directory. Also make a local copy of the *fvOptions/Make* directory.

```
mkdir --parents $WM_PROJECT_USER_DIR/applications/FGMFoam/src/fvOptions
cp -r $FOAM_SRC/fvOptions/{allSetOption,constraints,corrections,
↪  interRegionOptions,sources,Make}
↪  $WM_PROJECT_USER_DIR/applications/FGMFoam/src/fvOptions
```

Go to the user library directory for FGMFoam and modify *Make/files* for the two local version of turbulence model source files so that the libraries are created in the user library directory.

```
cd $LIB_FGM_SRC
sed -i 's/FOAM_LIBBIN/FOAM_USER_LIBBIN/g' TurbulenceModels/compressible/Make/files
↪  TurbulenceModels/turbulenceModels/Make/files
```

Again, change the name of the created library by adding 'FGMFoam' to the library name to distinguish it from the standard *fvOptions* library.

```
sed -i '1h;1!H;$!d;x;s/.*lib/&FGM/' fvOptions/Make/files
```

Also update the path variables in *Make/options* to access the customised libraries.

```
sed -i 's=LIB_LIBS.*=& \n    -L\$(FOAM_USER_LIBBIN) \\=' fvOptions/Make/files
```

Chapter 4

Tutorial FGMFoam

4.1 Pre-processing

This section covers the necessary setup needed to get the FGMFoam case running with chemistry.

4.1.1 Generate flamelet data

The script used to calculate 1D counter-flow diffusion flames based on an example Cantera script available online [12] is shown in Appendix A. The detailed reaction mechanism GRI-Mech 3.0 is used to calculate the species concentration and fluid parameters along 1D flames. The strain rate is increased after the calculation of the previous flamelet is finished. This is continued until extinction and results in approximately 50 separate flamelets at different strain rates.

The generated flamelet data is then prepared for the generation of the 4D table. As a first step, a python script shown in Appendix B reads the flamelet data and determines mixture fraction and progress variable at each point of the flamelet. The strain rate is determined from the velocity gradient. For the description of how mixture fraction Z and progress variable PV are calculated in this case, the reader is referred to Section 6.2.1 of L. Ma's PhD thesis [1]. In summary, the mixture fraction is selected to be defined as

$$Z = \frac{b - b_o}{b_f - b_o}, \tag{4.1}$$

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

$$b = 2\frac{Y_C}{M_C} + \frac{1}{2}\frac{Y_H}{M_H} - \frac{Y_O}{M_O}, \tag{4.2}$$

based on the elemental mass fractions of carbon, hydrogen and oxygen. The symbol M denotes the atomic weight of the respective element. The progress variable, PV, is selected by the author [1] to be

$$PV = 4\frac{Y_{H_2O}}{M_{H_2O}} + 2\frac{Y_{CO_2}}{M_{CO_2}} + \frac{1}{2}\frac{Y_{H_2}}{M_{H_2}} + \frac{Y_{CO}}{M_{CO}}, \tag{4.3}$$

and this definition is also used in the presented tutorial. These three equations are used in the postprocessing script for the 1D data, shown in Appendix B.

The processed flamelet results in *.csv* format can now be imported by a script to generate a flamelet generated manifold table of desired dimension, with appropriate parameters. For a Matlab sript to generate 4D tables with the tabulation parameters PV, Z, varPV, and varZ, please contact Likun Ma. To follow this tutorial, it is sufficient to copy the provided tables to the run directory. Note that these tables are only suitable for the simulation of certain methane/air flames at atmospheric pressure, with all other restrictions of the flamelet approach. The result of the tabulation process is then stored in a directory accessible for FGMFoam, in this tutorial case they are stored in the directory `$FOAM_RUN/02_table`. This directory contains 22 tables in OpenFOAM format, thereof 7 species. (Note that a reaction mechanisms with 47 species is used to generate the flamelet data. The number of provided tables is lower to reduce the file size in this tutorial.) The rest are tabulated thermo, combustion, and fluid properties. Also included are 6 tables (*Yb2I*, *YbWI*, *Yu2I*, *YuWI*, *YuYb*, *YWI*) necessary to determine the tabulation parameters scaledPV and variance of scaledPV for each timestep based on the progress variable PV and varPV.

4.1.2 Getting started

As in the previously described reactingFoam tutorial, copy the SandiaD_LTS tutorial case to the run directory. Using OpenFOAM v7,

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

4.1.3 Chemistry

In `constant/combustionProperties` at keyword *combustionModel* change from the EDC combustion model *EDC* to *FGMModel*. Also exchange the EDC coefficients for the varPV-switch of the FGMModel. The resulting *combustionProperties* file should be

```
combustionModel  FGMModel;
active  true;
FGMModelCoeffs
{
        useProgressVariableVariance true;//false;
}
```

The next necessary modification is to add the two files *tableProperties* and *PVtableProperties* to the constant directory. These two files are provided for convenience. The first part of each file specifies the definition of the mixture fraction, the interpolation type, the operating pressure and the path to the FGM tables. This is followed by a description of the grid points of the 4D FGM tables. In this case, the discretisation is 51 for Z and PV, and 5 for their variance. All four tabulation control parameters in an interval from 0 to 1.

```
mixtureFractionDefinition   "readFromTable";
interpolationType           linearInterpolation;
operatingPressure           1e5;
tablePath                   "$FOAM_RUN/TablePre1/";

varPV_param
5
(
0
...
1
)
PV_param
51
(
 0
...
1
)
varZ_param
5
(
0
...
1
)
Z_param
51
(
0
...
1
);
```

Similarly, for the interpolation of the progress variable *PVtableProperties* specifies the regular grid for the interpolation.

```
mixtureFractionDefinition   "readFromTable";
interpolationType           PVlinearInterpolation;
operatingPressure           1e5;
tablePath                   "$FOAM_RUN/TablePre1/";

varZ_param
5
(
0
...
1
)
Z_param
51
(
 0
...
1
);
```

Another modification is to switch off the radiation model, as this is not yet considered in the implementation of FGMFoam. *constant/fvOptions* should be empty apart from the header.

4.1.4 Boundary and initial conditions

The first step is to rename the field *0/T.orig* to *T*. However, note that the temperature is retrieved from the FGM table. 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*. The tablulated source term for the progress variable, 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.

Clean the 0 directory by removing all the files for the species mass fractions ($N2$, $CH4$ etc.), $Ydefault$, G, and mut.

4.1.5 Solution settings

Set the time integration scheme to Euler, and add the lines

```
    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;
```

in fvSchemes under divSchemes. At the end of the fvSchemes file, add

```
fluxRequired
{
    default         no;
    p;
}
```

Finally, replace the file fvSolution with

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

4.2 Running the code

Change the time step size to 5e-5 and the endTime to 0.1, where the solution has converged. Run *blockMesh* and then the FGM solver by typing *FGMFoam*. An Allrun script is provided to generate the mesh, run FGMFoam, and postprocess with FGMFoamPost.

4.3 Post-processing and ParaView

FGMFoam has its own postprocessing routine, FGMFoamPost, to look-up the tabulated species. The species which are looked-up depend on the list of species in the foamChemistryFile or CHEMKINFile. The selection and the path is specified in constant/thermophysicalProperties. The file constant/reactions includes only 7 species, for which tables are provided. Therefore make the following change to constant/thermophysicalProperties. Replace the last three lines with

```
CHEMKINFile "$FOAM_CASE/chemkin/reactions.inp";
CHEMKINThermoFile "$FOAM_CASE/chemkin/therm.dat";
CHEMKINTransportFile "$FOAM_CASE/chemkin/transportProperties";
```

Both chemkin files to which the paths are specified above are provided along with this report. The former is a dummy mechanism file and the latter the corresponding thermo file. The chemistry files are only used to determine the species for postprocessing, and do not affect the simulation result. Now run the postprocessing routine.

```
FGMFoamPost -latestTime
```

To assess the accuracy of the FGM solver, the OH-radical fields are compared between FGMFoam and reactingFoam. This comparison is shown on the left in Figure 4.1. On the right side, a comparison of the temperature fields given by the two different solvers is given. It can be observed that the quantitative difference between the predicted OH distribution is relatively high. However, the flame position is very similar, as demonstrated by the shape of the OH distribution and the temperature distribution. For cases with complex geometry and large range of flow length scales, the efficiency of the FGM model during runtime is more advantageous than for this tutorial case. In the provided material, a 2D flamelet generated manifold solver, FGSFoam is also included, along with an example case and the necessary tables. This solver is based on 2D tabulation as a function of mixture fraction and progress variable, and allows further speed-up.

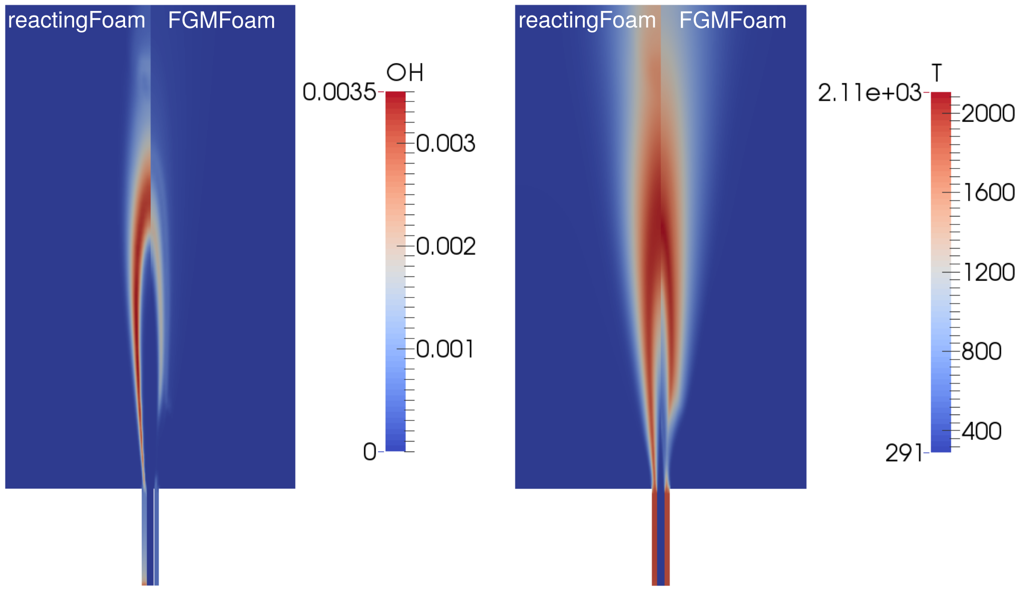

Figure 4.1: Comparison of results for the SandiaD_LTS tutorial case from reactingFoam and FGM-Foam.

Bibliography

[1] L. Ma, "Computational modelling of turbulent spray combustion", PhD, Delft University of Technology, 2016, ISBN: 978-94-6233-281-2.

[2] A. Lundström, "Simple gas phase reaction", in *Proceedings of CFD with OpenSource Software, 2008, Edited by Nilsson. H.*, 2008. DOI: `http://dx.doi.org/10.17196/OS_CFD#YEAR_2008`. [Online]. Available: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2007/AndreasLundstrom/reactingFoam.pdf`.

[3] C. Andersen and N. E. Nielsen, "Numerical investigation of a bfr using OpenFOAM", Aalborg Univsersity - Institute of Energy Technology, Tech. Rep., Jun. 2008. [Online]. Available: `https://projekter.aau.dk/projekter/files/14411784/Report.pdf`.

[4] P. P. Thummala, "Description of reactingTwoPhaseEulerFoam solver with a focus on mass transfer modeling terms", in *Proceedings of CFD with OpenSource Software, 2008, Edited by Nilsson. H.*, 2016. DOI: `http://dx.doi.org/10.17196/OS_CFD#YEAR_2016`. [Online]. Available: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016/PhanindraPrasadThummala/reactingTwoPhaseEulerFoam.pdf`.

[5] R. Barlow and J. Frank, *Piloted CH4/Air Flames C, D, E, and F; Release 2.1*, 2007. [Online]. Available: `http://www.sandia.gov/TNF/DataArch/FlameD/SandiaPilotDoc21.pdf`.

[6] H. Weller, *TDACChemistryModel: new chemistry model providing tabulation of dynamic adaptive chemistry*, 2016. [Online]. Available: `https://github.com/OpenFOAM/OpenFOAM-dev/commit/f2c263b9fd7c65e738b626280b8f2f3e7c1dadf8`.

[7] H. Chopkap Noume, V. Bomba, and M. Obounou, "Numerical Investigation of a Turbulent Jet Flame With a Compact Skeletal Mechanism", *Journal of Energy Resources Technology*, vol. 142, no. 3, Sep. 2019, 032206, ISSN: 0195-0738. DOI: `10.1115/1.4044556`. eprint: `https://asmedigitalcollection.asme.org/energyresources/article-pdf/142/3/032206/5425623/jert\_142\_3\_032206.pdf`. [Online]. Available: `https://doi.org/10.1115/1.4044556`.

[8] M. Bösenhofer, E.-M. Wartha, C. Jordan, and M. Harasek, "The eddy dissipation concept-analysis of different fine structure treatments for classical combustion", *Energies*, vol. 11, no. 7, 2018, ISSN: 1996-1073. DOI: `10.3390/en11071902`. [Online]. Available: `https://www.mdpi.com/1996-1073/11/7/1902`.

[9] B. Haddadi, C. Jordan, J. Nagy, *et al.*, *OpenFOAM basic training: Tutorial eleven, reactingFoam*, 2015. [Online]. Available: `https://www.cfd.at/sites/default/files/tutorials/2014_OFoam_Tut_Example%20Eleven.pdf`.

[10] H. Müller and L. Ma, *Extend-bazaar/solvers/combustion/flameletfoam*, 2014. [Online]. Available: `http://openfoamwiki.net/index.php/Extend-bazaar/solvers/combustion/flameletFoam`.

[11] M. Pfitzner, H. Müller, and F. Ferraro, "Implementation of a steady laminar flamelet model for nonpremixed combustion in LES and RANS simulations", Jun. 2013.

[12] *Diffusion_flame_batch.py*. [Online]. Available: `https://cantera.org/examples/python/onedim/diffusion_flame_batch.py.html`.

Appendix A

Python Cantera scripts to calculate 1D flamelets

```python
import cantera as ct
import numpy as np
import csv
import math
import os
class FlameExtinguished(Exception):
        pass
air = 'O2:0.21, N2:0.79'
fuel = 'CH4:1'
reaction_mechanism = 'gri30.xml' #'mech.cti'
gas = ct.Solution(reaction_mechanism)
f = ct.CounterflowDiffusionFlame(gas, width=1.)
f.transport_model = 'Multi'
f.P = 101325.
f.fuel_inlet.mdot = 0.01;
f.fuel_inlet.X = fuel;
f.fuel_inlet.T = 298.;
f.oxidizer_inlet.mdot = 0.025;
f.oxidizer_inlet.X = air;
f.oxidizer_inlet.T = 298.;
f.set_refine_criteria(ratio=3.0, slope=0.1, curve=0.2, prune=0.03)
temperature_limit_extinction = 700
def interrupt_extinction(t):
        if np.max(f.T) < temperature_limit_extinction:
                raise FlameExtinguished('Flame extinguished')
        return 0.
f.set_interrupt(interrupt_extinction)
# strain rate loop
strain_factor = 1.3 #increase strain rate by 30\% each step
exp_mdot_a = 1./2.  #coeff from Fiala and Sattelmayer 2014
data_directory = '01_flamelet_results/'
file_name = 'initial_solution.xlm'
n = 0
while (np.max(f.T) > temperature_limit_extinction or n==0):
        n += 1
        print('strain rate iteration', n)
        f = ct.CounterflowDiffusionFlame(gas, width=1.)
        f.transport_model = 'Multi'
        f.P = 101325.
        f.fuel_inlet.X = fuel;
        f.fuel_inlet.T = 298.;
        f.oxidizer_inlet.X = air;
        f.oxidizer_inlet.T = 298.;
        f.fuel_inlet.mdot = 0.01 * strain_factor ** (n*exp_mdot_a)
        f.oxidizer_inlet.mdot = 0.025 * strain_factor ** (n*exp_mdot_a)
        f.set_refine_criteria(ratio=3.0, slope=0.1, curve=0.2, prune=0.03)
        try:
                f.solve(loglevel=0, auto=True)
                file_name = 'strain_loop_' + format(n, '02d') + '.xml'
                f.save(data_directory + file_name, name='diff1D', loglevel=0,
                           description='Cantera version ' + ct.__version__ +
                           ', reaction mechanism ' + reaction_mechanism)
        except FlameExtinguished:
                print('Flame extinguished')
                break
        except ct.CanteraError as e:
                print('Error occurred while solving:', e)
                break
        print("Solved on {:d} points.".format(len(f.T)))
        print("T max {:.0f} K.".format(max(f.T)))
```

Appendix B

Python Cantera scripts to organise flamelet data

```
import cantera as ct
import numpy as np
import csv
import math
import os
import glob
import errno
def derivative(x, y):
        dydx = np.zeros(y.shape, y.dtype.type)
        dx = np.diff(x)
        dy = np.diff(y)
        dydx[0:-1] = dy/dx
        dydx[-1] = (y[-1] - y[-2])/(x[-1] - x[-2])
        return dydx
def computeStrainRates(f,Z):
    # Compute the derivative of axial velocity to obtain normal strain rate
    strainRates = derivative(f.grid, f.u)
    gradZ = derivative(f.grid, Z)
    return strainRates, gradZ
def computeBilger(composition):
    # Compute components of Bilger's formula
    gas_b = ct.Solution('gri30.xml')
    gas_b.X = composition
    bilger = \
    2.0*gas_b.elemental_mass_fraction('C')/gas_b.molecular_weights[gas_b.element_index('C')] +\
    0.5*gas_b.elemental_mass_fraction('H')/gas_b.molecular_weights[gas_b.element_index('H')] -\
    1.0*gas_b.elemental_mass_fraction('O')/gas_b.molecular_weights[gas_b.element_index('O')]
    return bilger
raw_directory = '01_flamelet_results/'
for name in glob.glob(raw_directory + '*.x*'):
        try:
                #with open(name) as g:
                #        pass
                print(name)
                reaction_mechanism = 'gri30.xml' #'mech.cti'
                gas = ct.Solution(reaction_mechanism)
                f = ct.CounterflowDiffusionFlame(gas, width=1.)
                name_only = name.rsplit('\\',1)[1] #delete directory in filename
                file_name = name_only
                f.restore(filename=raw_directory + file_name, name='diff1D', loglevel=0)
                air = 'O2:0.21, N2:0.79'
                fuel = 'CH4:1'
                b_fuel = computeBilger(fuel)
                b_air = computeBilger(air)
                print('Calculating Z...')
                b = []
```

```python
                for x in range(0, len(f.X[0,:])):
                        b.append(computeBilger(f.X[:,x]))
                Z = (b - b_air)/(b_fuel - b_air) # mixture fraction
                print('Calculating PV...')        # progress variable
                Y =\
                4.0*f.Y[gas.species_index('H2O')]/gas.molecular_weights[gas.species_index('H2O')] +\
                2.0*f.Y[gas.species_index('CO2')]/gas.molecular_weights[gas.species_index('CO2')] +\
                0.5*f.Y[gas.species_index('H2')]/gas.molecular_weights[gas.species_index('H2')] +\
                1.0*f.Y[gas.species_index('CO')]/gas.molecular_weights[gas.species_index('CO')]
                [strainRates, gradZ] = computeStrainRates(f,Z)
                a_max = max(strainRates) # maximum axial strain rate
                chi = 2*f.u*Z*gradZ # instantaneous scalar dissipation rate [1/s]
                alpha = f.thermal_conductivity/f.cp # thermal diffusivity * rho [kg/(ms)]
                gas.basis = 'molar'
                density_molar = f.density
                gas.basis = 'mass'
                density_mass = f.density
                molar_mass = density_mass/density_molar
                print('Preparing data...')
                data = np.transpose([\
                f.grid,\
                f.u,\
                f.T,\
                Z,\
                Y,\
                f.Y[gas.species_index('CO2')],\
                f.Y[gas.species_index('H2O')],\
                f.Y[gas.species_index('CO')],\
                f.Y[gas.species_index('H2')],\
                f.elemental_mass_fraction('C'),\
                f.elemental_mass_fraction('H'),\
                f.elemental_mass_fraction('O'),\
                f.net_production_rates[gas.species_index('CO2')]*\
                gas.molecular_weights[gas.species_index('CO2')],\
                f.net_production_rates[gas.species_index('H2O')]*\
                gas.molecular_weights[gas.species_index('H2O')],\
                f.net_production_rates[gas.species_index('CO')]*\
                gas.molecular_weights[gas.species_index('CO')],\
                f.net_production_rates[gas.species_index('H2')]*\
                gas.molecular_weights[gas.species_index('H2')],\
                f.density,\
                f.cp,\
                alpha,\
                f.viscosity,\
                molar_mass,\
                chi,\
                f.heat_release_rate,\
                f.h,\
                f.thermal_conductivity,\
                ])
                for k in range(0, len(f.Y[:,0])):
                        data = np.c_[data, np.transpose(f.Y[k,:])] #all 53 species
                print('Writing data...')
                data_directory = '02_processed_flamelet_results/'
                filename = 'post_' + name_only[:-4] + '.csv'
                try:
                        os.remove(filename)
                except OSError:
                        pass
                header = "**********************************************************\n"
                header += "1D counterflow diffusion flame\n"
                header += "fuel: " + fuel + "\n"
                header += "oxidiser: " + air + "\n"
                header += "p " + str(f.P) + " Pa"
                header += "T_in_fuel " + str(f.fuel_inlet.T) + "\n"
                header += "T_in_air " + str(f.oxidizer_inlet.T) + "\n"
                header += "**********************************************************\n"
```

```
            header += " z[m]                     u[m/s]                    T[K]
            ↪  Z                             PV[mol/g]            Y.CO2
            ↪  Y.H2O                  Y.CO                         Y.H2
            ↪  Y.C                       Y.H                     Y.O
            ↪  wCO2[kg/m^3s]  wH2O[kg/m^3s]  wCO[kg/m^3s]   wH2[kg/m^3s]         rho[kg/m^3]
            ↪  cp[J/kgK]      alpha[kg/ms]   mu[Pas]        Mbar[g/mol]    chi[1/s]
            ↪  hrr[W/m^3]     h[J/kg]        lambda[W/mK]   'H2'           'H'            'O'
            ↪  'O2'           'OH'           'H2O'          'HO2'           'H2O2'         'C'
            ↪  'CH'           'CH2'           'CH2(S)'      'CH3'          'CH4'          'CO'
            ↪  'CO2'           'HCO'         'CH2O'          'CH2OH'        'CH3O'        'CH3OH'
            ↪  'C2H'         'C2H2'          'C2H3'         'C2H4'         'C2H5'         'C2H6'
            ↪  'HCCO'          'CH2CO'       'HCCOH'          'N'            'NH'           'NH2'
            ↪  'NH3'           'NNH'          'NO'          'NO2'           'N2O'         'HNO'
            ↪  'CN'           'HCN'         'H2CN'          'HCNN'         'HCNO'         'HOCN'
            ↪  'HNCO'         'NCO'          'N2'            'AR'           'C3H7'        'C3H8'
            ↪  'CH2CHO'        'CH3CHO'"
            np.savetxt(data_directory + filename,data,fmt="%+.7e",header=header)
    except IOError as exc:
            if exc.errno != errno.EISDIR:
                    raise
```

Study questions

1. How is flame-turbulence interaction modelled in the EDC combustion model, one of the combustion models available in reactingFoam?
2. How are reaction kinetics incorporated into the solution process of reactingFoam?
3. What is the purpose of tabulated chemistry models?
4. What are the limitations for flamelet-based combustion models?
5. Which libraries need to be changed in order to successfully compile the combustion model library for the FGMFoam solver?
6. How is the viscosity modelled in the FGMFoam reactive flow solver?