Coupling OpenFOAM with Cantera for advanced diffusion modeling

Aleksi Rintanen

School of Engineering,
Aalto University,
Helsinki, Finland

January 22, 2025

Multicomponent reactive flows 1/2

The governing equations of multicomponent reactive flows include the complete Navier-Stokes equations (density, momentum, and energy) and species transport equations for each species.

OpenFOAM 12 has a solver module `multicomponentFluid`, which implements the following equation for the species transport using mass fractions Y_k as solution variables:

$$\frac{\partial}{\partial t}(\rho Y_k) + \nabla \cdot (\rho \mathbf{u} Y_k) = -\nabla \cdot \mathbf{j}_k + \dot{\omega}_k$$

The diffusive flux $-\nabla \cdot \mathbf{j}_k$ depends on the used model which is chosen at runtime. The species interact through the chemical source term $\dot{\omega}_k$, which is solved inside the chemistry model.

Multicomponent reactive flows 2/2

The energy equation can be solved for different solution variables, but here we consider the variant for specific sensible enthalpy h_s (`sensibleEnthalpy`) and the energy equation takes the following form:

$$\frac{\partial}{\partial t}(\rho h_s) + \nabla \cdot (\rho \mathbf{u} h_s) + \frac{\partial}{\partial t}(\rho K) + \nabla \cdot (\rho \mathbf{u} K) = \frac{\partial p}{\partial t} - \nabla \cdot \mathbf{q} + \dot{\omega}_T,$$

Here K is the specific kinetic energy of the fluid, $\dot{\omega}_T$ heat release due to the chemical reactions and $-\nabla \cdot \mathbf{q}$ the diffusive heat flux.

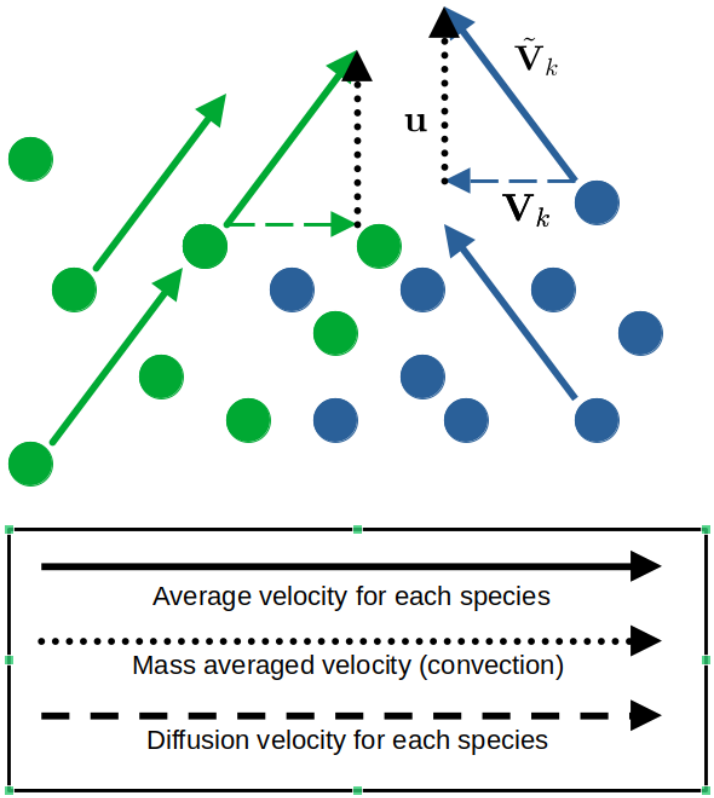
In this work we concentrate on how the diffusive fluxes $-\nabla \cdot \mathbf{j}_k$ and $-\nabla \cdot \mathbf{q}$ are modeled in OpenFOAM 12 and how it can be modeled more accurately.

Diffusion velocity

Diffusion velocity $\mathbf{V}_k$ is defined as the average speed of the species $\tilde{\mathbf{V}}_k$ relative to the mass averaged velocity $\mathbf{u}$

$$\mathbf{j}_k = \rho Y_k (\tilde{\mathbf{V}}_k - \underbrace{\sum_{j=1}^{N} Y_j \tilde{\mathbf{V}}_j}_{=\mathbf{u}}) = \rho Y_k \mathbf{V}_k$$

Also there exists also a definition for the diffusion velocity relative to molar averaged velocity



Heat flux and unity Lewis model 1/2

The diffusive flux $\mathbf{q}$ is calculated assuming Fourier's law:

$$\mathbf{q} = -\lambda \nabla T + \sum_{k=1}^{N} h_{s,k} \mathbf{j}_k,$$

A common simplification is to assume that mass diffusivities equal thermal diffusivity (unity Lewis assumption). The equations for $\mathbf{q}$ and $\mathbf{j}_k$ then simplify to:

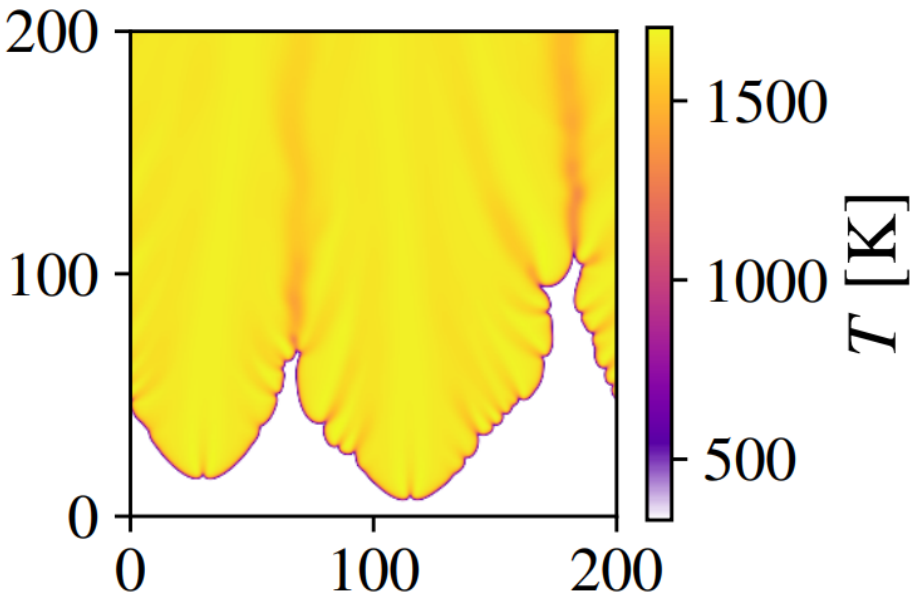
$$\mathbf{j}_k = -\alpha \nabla Y_k$$
$$\mathbf{q} = -\alpha \nabla h_s$$

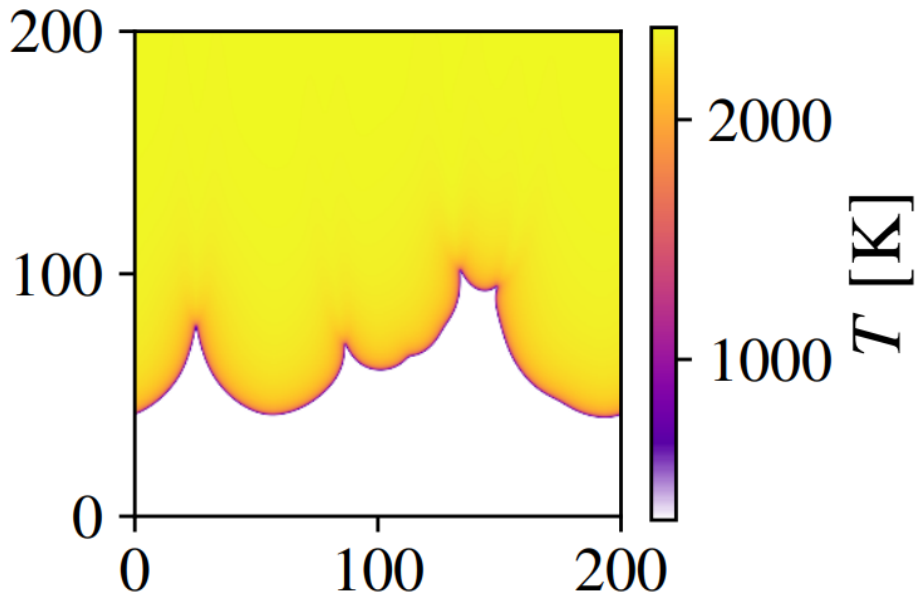
This is the default model used by OpenFOAM and also the only model until the introduction of more complex transport models in OpenFOAM 9.

Unity Lewis model 1/2

In many applications this assumption works very well, but in simulations involving hydrogen gas this might be inaccurate. Hydrogen gas is very light and has very high diffusivity. In the context of premixed hydrogen, the effects of predifferential diffusion can have large effect.

Here two 2D planar hydrogen flames are simulated using mixture-averaged formulation (left figure) and unity Lewis model (right figure). The difference in the flame characteristics is large.





Multicomponent diffusion model 1/2

Most accurate model is given by the multicomponent diffusion model, which is defined as

$$\mathbf{j}_k = \rho \frac{W_k}{\overline{W}} \sum_{j=1}^{N} W_j D_{kj} \left(\nabla X_j + (X_j - Y_j) \frac{1}{p} \nabla p \right) - D_k^T \frac{1}{T} \nabla T$$

The diagonal of the ordinary multicomponent diffusion coefficient matrix D_{jk} is zero, which can pose some numerical challenges. An alternative formulation is the Maxwell-Stefan model, where diffusion velocities are related to the field gradients as

$$\nabla X_k = \sum_{j=1}^{N} \frac{X_k X_j}{\mathcal{D}_{kj}} (\mathbf{v}_j - \mathbf{v}_k) + (Y_k - X_k) \left(\frac{\nabla p}{p} \right) + \sum_{j=1}^{N} \frac{X_k X_j}{\rho \mathcal{D}_{kj}} \left(\frac{D_j^T}{Y_j} - \frac{D_k^T}{Y_k} \right) \frac{\nabla T}{T}$$

The diffusion due to pressure gradient is often neglected.

Thermal diffusion

- Thermal diffusion, also known as Soret effect, is diffusion due to temperature gradient, represented by the last term in the multicomponent model $-D_k^T \frac{1}{T} \nabla T$
- The evaluation of the thermal diffusion coefficients D_k^T in the multicomponent model requires solving a linear system of size $3N \times 3N$, where N is the number of species.
- Multicomponent/Maxwell-Stefan model without Soret diffusion involves solving of $N \times N$ size system

Mixture-averaged model

The diffusive mass flux is related to the mass fraction gradient as

$$\mathbf{j}_k = -\rho D_k \nabla Y_k,$$

This mixture-averaged diffusion coefficient D_k can be obtained from the Maxwell-Stefan equations by neglecting thermal diffusion, diffusion due to pressure gradient and assuming that the diffusion velocities in the formulation are equal for $j \neq k$

$$D_k = \left(\sum_{j \neq k} \frac{X_j}{\mathcal{D}_{jk}} + \frac{X_k}{1 - Y_k} \sum_{j \neq k} \frac{Y_j}{\mathcal{D}_{jk}} \right)^{-1},$$

This is gives a very good approximation of the multicomponent model
The thermal diffusion term can also be included in the mixture-averaged model

ThermophysicalTransportModels

- In OpenFOAM terminology `ThermophysicalTransportModels` are models that implements $-\nabla \cdot \mathbf{j}_k$ and $-\nabla \cdot \mathbf{q}$ terms in the governing equations
- These models were introduced in OpenFOAM 9
- Until now those are only present in the foundation versions
- The idea is the same as in `momentumTransportModels`:
  - Solver independent models, no code duplication
  - Model is selected at runtime, which makes switching them easy
  - Separate models for laminar and LES/RAS simulations
  - Used in the equations similarly (`turbulence->divDevTau(U)` vs `thermophysicalTransport->divj(Yi)`)

Selecting the model

User defines the model and model coefficients in the `thermophysicalTransportDict` dictionary file in the constant directory. When a laminar turbulence model is used, the model name is looked up under the laminar subdictionary. For RAS/LES simulations, their corresponding subdictionaries are used.

An example of `thermophysicalTransportDict` file

```
laminar
{
    model           unityLewisFourier;
}
```

Selecting the model

- There are several base instances for thermophysical transport models similarly as there are for momentum transport models (icompressible, compressible etc.)
- Each of these instances have three base models in their runtime tables: laminar, RAS, and LES.
- Each base model has its own selector and runtime table for the actual models.
- The names are constructed from two parts: suffix `Fourier` means that the heat flux is propotional to the temperature gradient
- `EddyDiffusivity` is the LES/RAS counterpart and here the turbulent closure is modelled by using eddy diffusivity concept, where additional diffusion is introduced by assuming a relation to the turbulent viscosity ν_t as

$$\alpha_{\text{eff}} = \alpha + \frac{\nu_t}{Pr_t}$$

$$D_{k\text{eff}} = D_k + \frac{\nu_t}{Sc_t},$$

Available models in OpenFOAM-12

Available models for the `fluidMulticomponentThermo`, which is used by the reactive solver `multicomponentFluid`

Model	$\mathbf{j}_k$	$\mathbf{q}$	Type
`Fourier`[1]	-	$-\lambda \nabla T$	laminar
`unityLewisFourier`	$-\frac{\lambda}{c_p} \nabla Y_k$	$-\frac{\lambda}{c_p} \nabla h_s$	laminar
`FickianFourier`	$-D_k \nabla Y_k$	$-\frac{\lambda}{c_p} \nabla T + \sum_k h_{sk} \mathbf{j}_k$	laminar
`MaxwellStefanFourier`	Maxwell-Stefan Eq.[2]	$-\lambda \nabla T + \sum_k h_{sk} \mathbf{j}_k$	laminar
`eddyDiffusivity`[1]	-	$-(\lambda + \frac{\nu_t}{Pr_t}) \nabla T$	RAS/LES
`unityLewisEddyDiffusivity`	$-(\frac{\lambda}{c_p} + \frac{\nu_t}{Pr_t}) \nabla Y_k$	$-(\frac{\lambda}{c_p} + \frac{\nu_t}{Pr_t}) \nabla h_s$	RAS/LES
`nonUnityLewisEddyDiffusivity`	$-(\frac{\lambda}{c_p} + \frac{\nu_t}{Sc_t}) \nabla Y_k$	$-(\frac{\lambda}{c_p} + \frac{\nu_t}{Pr_t}) \nabla h_s$	RAS/LES
`FickianEddyDiffusivity`	$-(D_k + \frac{\nu_t}{Sc_t}) \nabla Y_k$	$-(\frac{\lambda}{c_p} + \frac{\nu_t}{Pr_t}) \nabla T + \sum_k h_{sk} \mathbf{j}_k$	RAS

[1] Supports single component systems only

[2] Assumes $\nabla p = 0$

Coupling Cantera with OpenFOAM 1/2

Cantera can be installed from pre-compiled binaries or can be compiled from the source code. Here instructions are given for Ubuntu 22.04 binaries. Start by adding Cantera's Personal Package Archive (PPA) to sources:

```
sudo apt install software-properties-common
sudo apt-add-repository ppa:cantera-team/cantera
```

Cantera (C++ version) with headers can then be installed with:

```
sudo apt install libcantera-dev
```

Instructions for the compiling Cantera from source code can be found from the project report

Coupling Cantera with OpenFOAM 2/2

Settings for compilation and linking of OpenFOAM programs/libraries utilizing Cantera can be automated using `pkg-config` tool, where
`pkg-config --cflags cantera` gives the compilation flags and
`pkg-config --libs cantera` linking flags.

Example of OpenFOAM's options file

```
EXE_INC = \
    $(shell pkg-config --cflags cantera)

LIB_LIBS = \
    -lyaml-cpp \
     $(shell pkg-config --libs cantera)
```

Using Cantera from OpenFOAM 1/2

The basic Cantera object is a solution object that can be created from a chemical mechanism file and its path `mech` is given as an argumen

```
1 Cantera::shared_ptr<Cantera::Solution> solution = Cantera::newSolution(mech);
```

Since the species can have different order in Cantera, an indexing list `List<label> speciesIndex_` is used as a mapping between OpenFOAM and Cantera species indicies.

Cantera uses the standard std-containers and mapping can be obtained looping over the specie names.

Using Cantera from OpenFOAM 2/2

Constructing species mapping between Cantera and OpenFOAM

```
const std::vector<std::string> & speciesNames = solution_->thermo()->
    speciesNames();
forAll(Y, i)
{
    const std::vector<std::string>::const_iterator iter =
        std::find(speciesNames.begin(), speciesNames.end(), Y[i].name());
    if(iter == speciesNames.end())
    {
        FatalErrorInFunction
            << "Specie " << Y[i].name() << " not found in Cantera mechanism."
            << exit(FatalError);
    }
    speciesIndex_[i] = std::distance(speciesNames.begin(), iter);
}
```

Creating a new thermophysical transport model

A new transport model is created by cloning `FickianFourier` class and file `fluidMulticomponentThermophysicalTransportModels.C` with its `Make` directory.

Copying the needed files

```
cp -r $FOAM_SRC/ThermophysicalTransportModels/fluidMulticomponentThermo .
cp -r $FOAM_SRC/ThermophysicalTransportModels/fluid/laminar/FickianFourier .
```

The FickianFourier class is then renamed into `CanteraMixtureAveraged` and the content of the `fluidMulticomponentThermophysicalTransportModels.C` file is modified to only contain an instance of the `CanteraMixtureAveraged` class:

fluidMulticomponentThermophysicalTransportModels.C

```
#include "fluidMulticomponentThermophysicalTransportModels.H"

#include "CanteraMixtureAveraged.H"
makeLaminarThermophysicalTransportModel(CanteraMixtureAveraged);
```

Creating a new thermophysical transport model

- Macro `makeLaminarThermophysicalTransportModel` will create an instance of the model using the template parameters defined in the `fluidMulticomponentThermophysicalTransportModels.H` file
- It also adds the class the the laminar runtime table under this particular `fluidMulticomponentThermo` class
- After chancing the installation path and library path in `Make/files` file, a new model can be successfully compiled

Modifications to the code

- All the code related to calculation of diffusion coefficients in member function `updateDm()` is replaced
- A boolean flag `soretEnabled_` is added as a private class member to indicate whether to include the computations for the Soret effect
- Diffusion coefficients are evaluated using Cantera's `getMixDiffCoeffs` method
- Thermal diffusion coefficients are calculated using method `getThermalDiffCoeffs`

Calculating diffusion coefficients 1/2

- First, the gas mixture is set to desired properties
- Then the method for calculating the diffusion coefficients is called
- This procedure is repeated for all the cells in the internal field
- Similar procedure is then performed for the boundary fields

Evaluation of diffusion coefficients

```
Cantera::shared_ptr<Cantera::ThermoPhase> gas = solution_->thermo();
scalarField Ys(Y.size());
scalarField Dms(Y.size());
scalarField DTs(Y.size());
scalar epsilon = 1-Y.size()*SMALL;
forAll(T,celli)
{
    forAll(Y,i)
    {
        if(Y[i][celli] > epsilon)
        {
            Ys[speciesIndex_[i]] = epsilon;
        }
        else {
            Ys[speciesIndex_[i]] = Y[i][celli] + SMALL;
        }
```

Calculating diffusion coefficients 2/2

```
17 Cantera::shared_ptr<Cantera::ThermoPhase> gas = solution_->thermo();
18
19     }
20     gas->setState_TPY(T[celli], p[celli], Ys.begin());
21     solution_->transport()->getMixDiffCoeffs(Dms.begin());
22     forAll(Y,i) {
23         Dm_[i][celli] = Dms[speciesIndex_[i]];
24     }
25     if(soretEnabled_)
26     {
27         solution_->transport()->getThermalDiffCoeffs(DTs.begin());
28         forAll(Y,i) {
29             DT_[i][celli] = DTs[speciesIndex_[i]];
30         }
31    }
32 }
```

Since the Eq. (9) is undefined at $Y_k = 1$ and the denominator flips its sign when $Y_k > 1$, a small value is added to the all the species and Y_k is restricted such that $\max Y_k < 1$ to prevent numerical problems in single component regions.

Correction velocity approach

- Summing together the transport equations for species should lead to the continuity equation, which is not satisfied by the mixture-averaged model
- In other words, mixture-averaged model does not conserve mass directly, since the diffusive mass fluxes do not cancel out completely
- The current OpenFOAM's mixture-averaged model `Fickian` uses an approach were all the errors in mass fluxes are dumped into inert species.
- A more accurate approach is the correction velocity approach

$$\mathbf{j}'_k = \mathbf{j}_k - Y_k \sum_{j=1}^{N} \mathbf{j}_k$$

- Now $\sum_{k=1}^{N} \mathbf{j}'_k$ is always zero.

Correction velocity approach

This is implemented as a new member function `correctJc()` and called in `predict()` function:

Calculation of the correction flux

```
template<class laminarThermophysicalTransportModel>
void CanteraMixtureAveraged<laminarThermophysicalTransportModel>::correctJc()
    const
{

   const PtrList<volScalarField>& Y = this->thermo().Y();
   const volScalarField& T = this->thermo().T();
   surfaceScalarField gradLogT = fvc::snGrad(T)/fvc::interpolate(T);
   Jc_ *= scalar(0);
   forAll(Y, i)
   {
        Jc_ -= fvc::interpolate(this->alpha()*this->DEff(Y[i]))*fvc::snGrad(Y[i
    ])
             + fvc::interpolate(DT_[i])*gradLogT;
   }

}
```

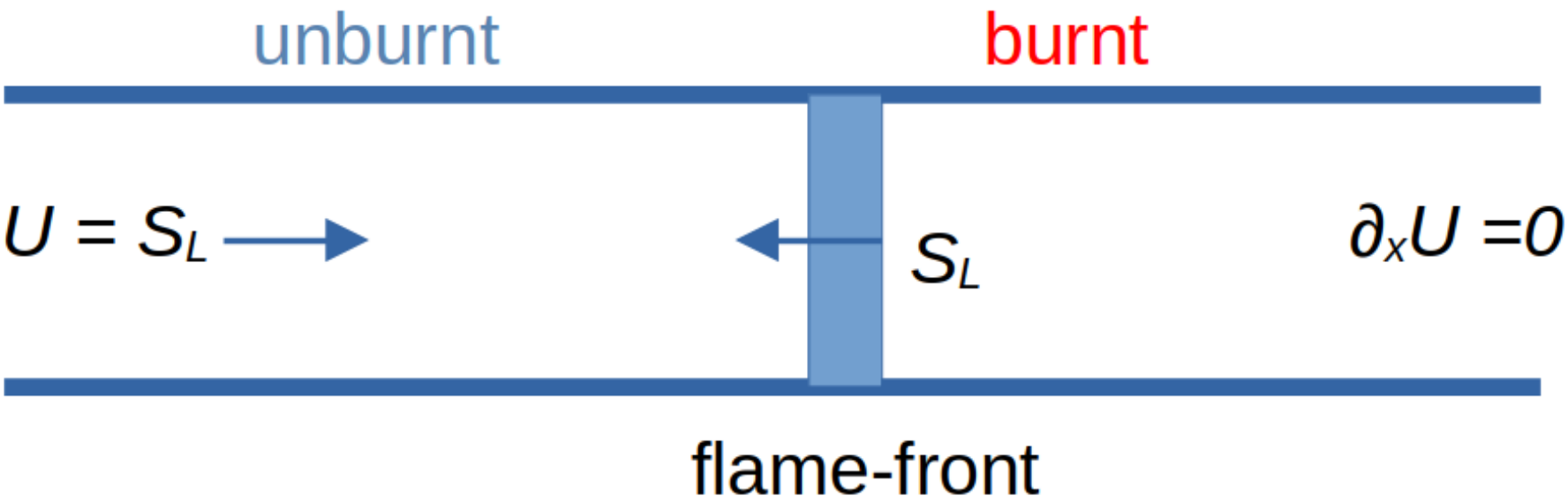
Correction velocity approach

The correction flux is added to the species flux implicitly and the divergence of $\mathbf{j}_k$ is then given as

```
return unityLewisFourier<laminarThermophysicalTransportModel>::divj(Yi)
           - fvc::div
             (
                 fvc::interpolate
                 (
                   DT_[this->thermo().specieIndex(Yi)]
                 )
                *fvc::snGrad(T)/fvc::interpolate(T)
                *T.mesh().magSf()
             )
            - fvm::div(Jc_*T.mesh().magSf(),Yi,"div(phi,Yi_h)");
```

Freely propagating 1D flame 1/2

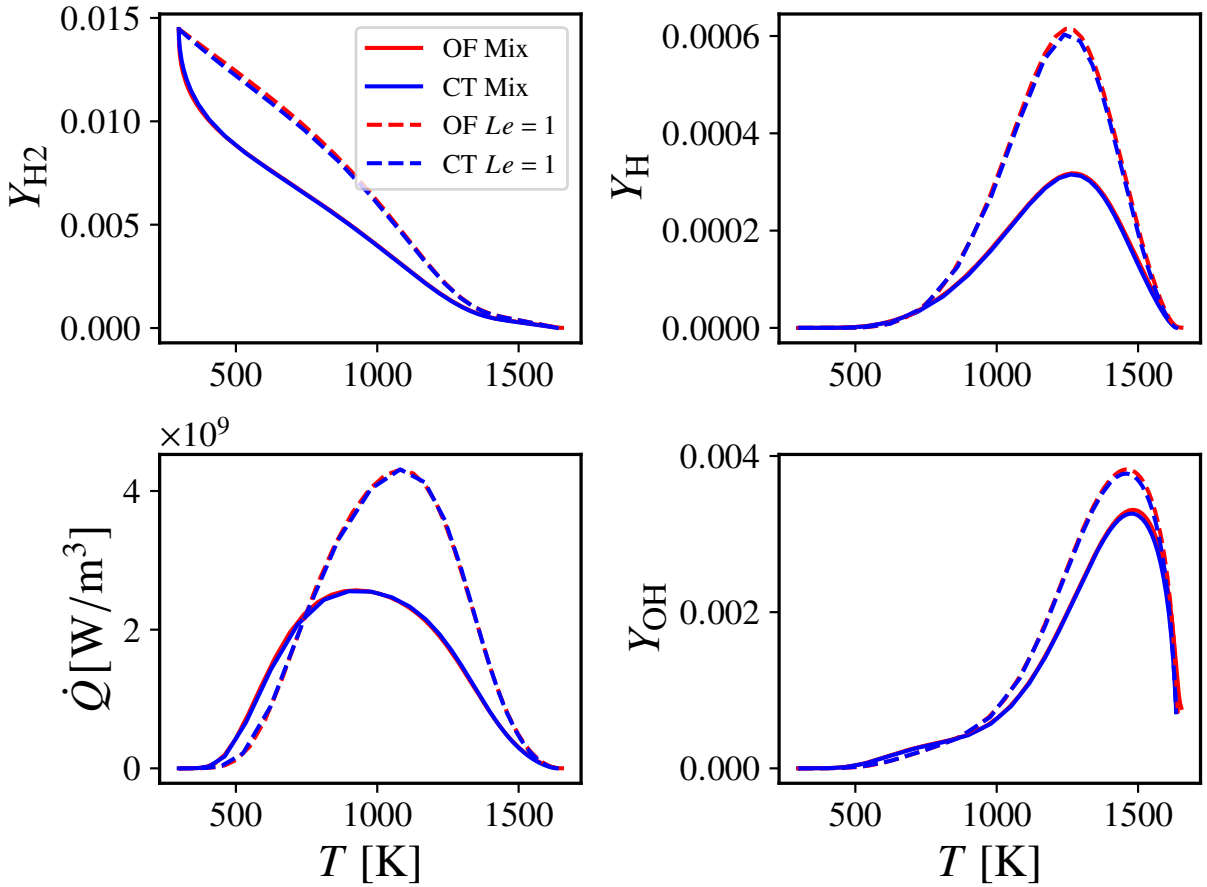
- For validation, a lean freely propagating planar hydrogen flame is considered
- A mixture of hydrogen and air is ejected into the domain from the left boundary (inlet) at the same velocity as the flame is propagating to the left (S_L)
- Equivalence ratio of the unburnt mixture is set to 0.5
- Temperature and pressure are set to ambient conditions



Freely propagating 1D flame 2/2

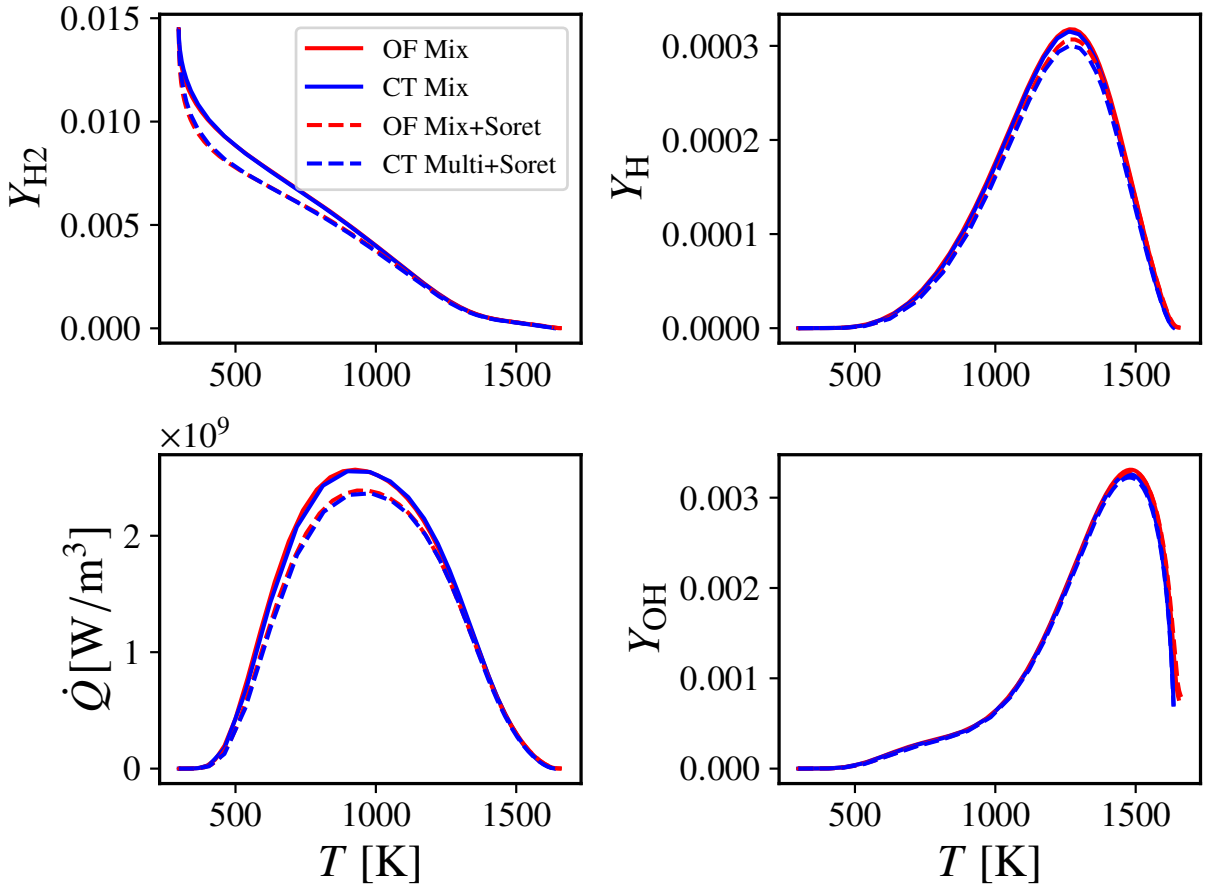
- Cantera can solve steady-state 1D freely propagating flames, which can be used to verify the implemented transport models.
- Cantera have three different transport models implemented: unity-Lewis, mixture-averaged and multicomponent, of which the last one can have the Soret effect enabled.
- The same uniformly spaced grid is used in both software, where cell spacing is defined such that there are 20 cells per laminar flame thickness.

Results 1/2



Comparison of Cantera and OpenFOAM solutions for 1D freely propagating flame using unity-Lewis and mixture-averaged transport.

Results 2/2



Comparison of Cantera and OpenFOAM solutions for 1D freely propagating flame with Soret effect.

Discussion

- The profiles agree very well and at the same time show the importance of accurate diffusion modeling in case of hydrogen flames.
- The solution from unity-Lewis model differs a lot from the mixture-averaged solution.
- Cantera does not allow using the combination of mixture-averaged transport and Soret effect and therefore the multicomponent transport model is used
- The reported differences of multicomponent and mixture-averaged diffusion model are small in the literature, and therefore the results are comparable

Tutorial

- The validation case is shipped along with the source code of the library
- The domain is initialized from a 1D freely propagating flame solution, which is generated using Cantera with unity-Lewis model
- This profile is used for all the cases and all the cases are run for 10 flame times to get the steady-state profiles.
- Also the python code for plotting is provided

Conclusions

- Cantera software was coupled with OpenFOAM, and a new mixture-averaged diffusion model with Soret effect was developed
- Additionally more robust correction velocity approach was deployed to balance the imbalance in mass fluxes
- An overview and implementation of the existing thermophysical transport models in OpenFOAM were given
- The model was validated by comparing the results between OpenFOAM and Cantera for one-dimensional freely propagating flame
- As a future work, more lightweight and less complex thermal diffusion models could be developed for OpenFOAM, which would neither require solving of linear system nor coupling with Cantera.

Thank you for your attention!

Questions?