Turbulence-chemistry interaction in OpenFOAM and how to implement a dynamic PaSR model for LES of turbulent combustion

Presenter: Arvid Åkerblom
Art: Moa Bergman
December 7, 2022
LES of turbulent combustion

- Large Eddy Simulation (LES) is a method for simulating unsteady turbulent flows.
- Large turbulent structures (~80% of kinetic energy) are resolved while smaller structures are modeled.
- In reactive flow, a transport equation is solved for the mass fraction $\tilde{Y}_i$ of each species $i$.

$$\frac{\partial \rho \tilde{Y}_i}{\partial t} + \nabla \cdot (\rho \tilde{Y}_i \tilde{v}) = \nabla \cdot (D_i \nabla \tilde{Y}_i) + \tilde{\omega}_i - \nabla \cdot b_i,$$

- Time derivative
- Convective term
- Diffusive term
- Species production rate
- Sub-grid source term
The filtering problem

- LES uses $\overline{\omega_i}$, the filtered reaction rate of species $i$
- Problem: How to compute $\overline{\dot{\omega}_i}$ if all the complicated subgrid chemistry is unresolved?
- My impression after speaking with colleagues:
  - There is no simple way to compute $\overline{\dot{\omega}_i}$
  - The underlying assumptions of many existing models are very questionable
  - We should all be scared and confused
Some turbulence-chemistry interaction models (simplified)

- In the **Perfectly Stirred Reactor (PSR)** model, each mesh cell is a homogeneous reactor. 
  \[ \dot{\omega}_{i,PSR} = \dot{\omega}_i \]

- The Eddy Dissipation Concept (EDC) model assumes that 
  \[ \dot{\omega}_i \approx \gamma^* \dot{\omega}_{i,PSR} \]. Simply put, turbulence causes reactions to only take place in localized, anisotropic pockets. The volume fraction \( \gamma^* \) of these pockets is determined from turbulence. Note: \( 0 \leq \gamma^* \leq 1 \).

- The Fractal Model (FM) is similar to EDC, but computes \( \gamma^* \) based on a cascade model.

- (Flamelet models can also be used to get around the \( \dot{\omega}_i \) problem, but in a different way.)
The Partially Stirred Reactor (PaSR) model (also simplified)

- The Partially Stirred Reactor (PaSR) model also assumes that $\bar{\omega}_i \approx \gamma^* \bar{\omega}_{i,PSR}$, but attempts to take both turbulence and chemistry into account when estimating $\gamma^*$:

$$\gamma^* = \frac{\tau_c}{\tau_c + \tau^*}$$

- $\tau_c$ is the chemical time scale
- $\tau^*$ is the mixing time scale
- Simply put: if turbulent mixing is fast relative to chemistry, reactions take place in a well-mixed environment.
Turbulence-chemistry interaction in OpenFOAM

- The laminar (a.k.a. PSR), EDC and PaSR models are implemented as classes in OpenFOAM.
- These inherit from ChemistryCombustion, which gives them access to the chemistry model

```
  CombustionModel<ReactionThermo>
   
  ChemistryCombustion<ReactionThermo>
       
  laminar<ReactionThermo>
       
  PaSR<ReactionThermo>  EDC<ReactionThermo>
```
Turbulence-chemistry interaction in OpenFOAM

The model classes contain three crucial functions.

```cpp
// Correct combustion rate
virtual void correct();

// Fuel consumption rate matrix.
virtual tmp<fvScalarMatrix> R(volScalarField& Y) const;

// Heat release rate [kg/m/s^3]
virtual tmp<volScalarField> Qdot() const;
```
The PaSR class has the following definitions for $R$ and $Q_{dot}$.

```cpp
template<class ReactionThermo>
Foam::tmp<Foam::fvScalarMatrix>
Foam::combustionModels::PaSR<ReactionThermo>::R(volScalarField& Y) const
{
    return kappa_*laminar<ReactionThermo>::R(Y);
}

template<class ReactionThermo>
Foam::tmp<Foam::volScalarField>
Foam::combustionModels::PaSR<ReactionThermo>::Qdot() const
{
    return tmp<volScalarField>
    {
        new volScalarField
        {
            this->thermo().phasePropertyName(typeName + ":Qdot"),
            kappa_*laminar<ReactionThermo>::Qdot()
        };
    }
}
The correct function calculates $\gamma^*$ (here referred to as kappa_). The crucial step is the following loop.

```csharp
forAll(epsilon, i)
{
    const scalar tk = 
        Cmix_*sqrt(max(muEff[i]/rho[i]/(epsilon[i] + SMALL), 0));

    if (tk > SMALL)
    {
        kappa_[i] = tc[i]/(tc[i] + tk);
    }
    else
    {
        kappa_[i] = 1.0;
    }
}
```

\[ \gamma^* = \frac{\tau_c}{\tau_c + \tau^*} \]

\[ \tau^* = \tau_K = \sqrt{\nu/\epsilon} \]
Computing the time scales

- So, how do we compute $\tau_c$ and $\tau^*$?
- In the code we just saw, $\tau^* = \tau_K = \sqrt{v/\varepsilon}$
- According to Sabelnikov & Fureby (2013), $\tau^* = \sqrt{\tau_K \tau_\Delta}$ and $\tau_\Delta = \Delta/v'$ (time scale of subgrid stretch). This is a better fit for the anisotropic shape of reactive fine structures.
- $\tau_c$ is a more complicated story. A review article from 2020 lists 9 different ways to compute it (Wartha et al. (2021)) and I know of at least one more.
Chemical time scale in OpenFOAM

OpenFOAM uses the following expression for $\tau_c$:

$$\tau_c = \sum_{j=1}^{n_R} \frac{c_{tot}}{\sum_{i=1}^{N_{s,RHS}} \nu_{i,j} k_{f,j}}$$

In words: "Each reaction contributes the total molar concentration $c_{tot}$ divided by the production rate [kmol/m$^3$/s] of all species on its Right-Hand Side (RHS). The chemical time scale is the sum of all such contributions."
Problems

- Using $\tau^* = \tau_K$ does not take anisotropy into account
- The OpenFOAM method for computing $\tau_c$ is not made with PaSR in mind
- Why is only the RHS of the reactions considered? What does this mean for reversible reactions?
- All methods I've seen compute a single $\tau_c$ and then use that to scale all reactions equally. This is a bit strange, as reactions can have wildly different time scales
- Using a single $\tau_c$ probably makes sense when you only have 1-4 reactions
- Nowadays we use chemical mechanisms with 200-300 reactions though…
A new model

To tackle these problems, I propose a new model: foxPaSR.

- Computes separate chemical time scales for all reactions
- $\tau^* = \sqrt{\tau_K \tau_\Delta}$
- Takes reversible reactions into account
- Suitable for reaction mechanisms of any size
- Does not require additional parameters
The foxPaSR algorithm

In each cell, at each time step:
1. Compute $\tau^* = \sqrt{\tau_k \tau_\Delta}$.
2. Use the PSR model to compute the rates of all reactions.
3. For each reaction $j$, first identify if the reverse rate is bigger than the forward rate. If so, the reverse reaction is considered to have its reactants on the RHS.
4. For each reaction $j$, identify the reactant species $i$ with the shortest residence time, defined as $\tilde{Y}_i \bar{\rho} / \nu_r,i k_r$. This residence time is chosen as $\tau_c$ for the reaction.
5. For each reaction $j$, compute $\gamma_j^* = \frac{\tau_{c,j}}{\tau_{c,j} + \tau^*}$.
6. Compute the filtered reaction rates by summing the contributions of all reactions:
   $$\tilde{\omega}_i = \sum_{j=1}^{n_R} \gamma_j^* \tilde{\omega}_{i,j,PSR}$$
Implementing foxPaSR

- foxPaSR is a heavily modified version of the existing PaSR class.
- Production rates are modified inside the correct function, while the R and Qdot functions just return the already modified rates.

```cpp
template<class ReactionThermo>
Foam::tmp<Foam::fvScalarMatrix>
Foam::combustionModels::foxPaSR<ReactionThermo>::R(volScalarField& Y) const
{
    return laminar<ReactionThermo>::R(Y);
}
```
Implementing foxPaSR

foxPaSR has three new members:

- \( Y_\)_, the species mass fractions
- \( \text{reactions}_\)_, the list of reactions
- \( \text{modRR}_\)_, the locally modified production rates of all species

```cpp
Y_(this-> thermo().composition(). Y()),
reactions_
{
    dynamic_cast< const_ reactingMixture<gasHTermoPhysics>& >(this-> thermo())
},
modRR_( this-> chemistryPtr_-> nSpecie())
```
Implementing foxPaSR

The correct function consists of two loops:

- an outer loop, over the list of reactions
- an inner loop, over all cells

```cpp
forAll(reactions_, ri)
{
    label refSpecies = reactions_[ri].lhs()[0].index;

    const scalar& refW =
        this->thermo().composition().W(refSpecies);

    scalarField refRR =
        this->chemistryPtr_->calculateRR(ri, refSpecies)/refW;

    forAll(epsilon, celli)
    {
        // To be filled in...
    }
}
```

For reaction $r_i$, the variable $refRR$ represents the rate at which kmol are transferred from the LHS to the RHS. It can be negative.
Implementing foxPaSR

In each cell, a residence time for each reactant species is computed.

\[ tc_{\text{Species}} = \text{mag} (\rho_0 \cdot C \cdot Y_{[i]} [celli] / (\text{refRRC} \cdot W \cdot \text{stoichCoeff})) \]

The shortest of these is chosen as the chemical time scale.

```cpp
if (tcSpecies < tc) {
  tc = tcSpecies;
}
```
Implementing foxPaSR

The mixing time scale is also computed.

\[ \tau_K = \sqrt{\frac{v}{\varepsilon}} \]
\[ \tau_\Delta = \frac{\Delta}{v'} \]
\[ \tau^* = \sqrt{\tau_K \tau_\Delta} \]
Implementing foxPaSR

The reaction-specific volume fraction $\gamma_j^*$ (kappa) is then computed following the standard PaSR approach. If the mixing is very fast or the chemistry very slow, $\gamma_j^* = 1$ as in the PSR model.

```c
if (tm > SMALL && tc < GREAT)
{
    kappa = tc/(tc + tm);
}
else
{
    kappa = 1.0;
}
```
Implementing foxPaSR

The scaled production rates in reaction $r_i$ are then added to the total modified production rates.

```cpp
modRR_[i][celli] -=
    kappa*refRRC*W*stoichCoeff;
```

When all reactions have been looped over, the production rates initially computed by PSR are replaced with the modified production rates.

```cpp
forAll(modRR_, i)
{
    this->chemistryPtr_->RR(i) = modRR_[i];
}
```

The modified production rates will now be used in the energy and species transport equations.
Some simple results

- Premixed n-heptane flame
- Pitz-Daily geometry with side walls
- Coarse-grained LES with LDKM

- $\text{Re} \approx 50,000$
- $\phi = 0.75$
- 340,000 hex cells
References

Thanks for listening!