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

cfdfox.com

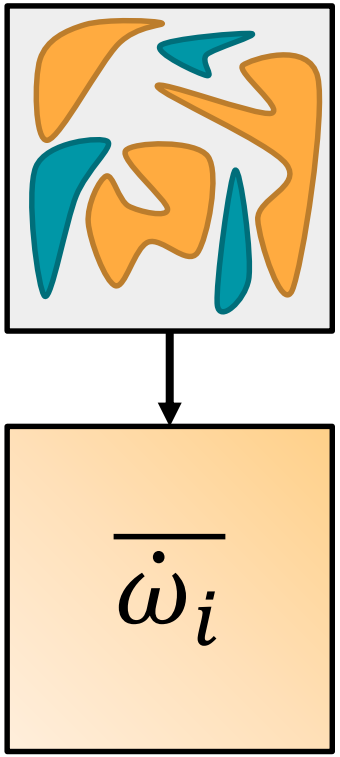
LES of turbulent combustion

- 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

$$\underbrace{\frac{\partial \overline{\rho}\tilde{Y}_i}{\partial t}}_{\text{Time derivative}} + \underbrace{\nabla \cdot (\overline{\rho}\tilde{Y}_i\tilde{\mathbf{v}})}_{\text{Convective term}} = \underbrace{\nabla \cdot (D_i \nabla \tilde{Y}_i)}_{\text{Diffusive term}} + \underbrace{\overline{\dot{\omega}_i}}_{\text{Species production rate}} - \underbrace{\nabla \cdot \mathbf{b_i}}_{\text{Sub-grid source term}},$$

The filtering problem

- LES uses $\overline{\dot{\omega}_i}$, 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. $\overline{\dot{\omega}_{i,PSR}} = \dot{\omega}_i$
- The Eddy Dissipation Concept (EDC) model assumes that $\overline{\dot{\omega}_i} \approx \gamma^* \overline{\dot{\omega}_{i,PSR}}$. Simply put, turbulence causes reactions to only take place in localized, anisotropic pockets. The volume fraction γ^* of these pockets is determined from turbulence. Note: $0 \leq \gamma^* \leq 1$.
- The Fractal Model (FM) is similar to EDC, but computes γ^* based on a cascade model.
- (Flamelet models can also be used to get around the $\overline{\dot{\omega}_i}$ problem, but in a different way.)

The Partially Stirred Reactor (PaSR) model (also simplified)

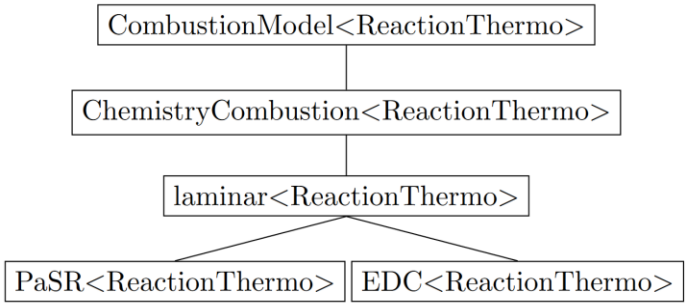
- The Partially Stirred Reactor (PaSR) model also assumes that $\overline{\dot{\omega}_i} \approx \gamma^* \overline{\dot{\omega}_{i,PSR}}$, but attempts to take **both turbulence and chemistry** into account when estimating γ^*:

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

- τ_c is the **chemical time scale**
- τ^* 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



Turbulence-chemistry interaction in OpenFOAM

The model classes contain three crucial functions.

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

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

//- Heat release rate [kg/m/s3]
virtual tmp<volScalarField> Qdot() const;
```

PaSR in OpenFOAM

The PaSR class has the following definitions for `R` and `Qdot`.

```
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()
        )
    );
}
```

PaSR in OpenFOAM

The `correct` function calculates γ^* (here referred to as kappa_).
The crucial step is the following loop.

```
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;
    }
}
```

$\tau^* = \tau_K = \sqrt{\nu/\varepsilon}$

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

Computing the time scales

- So, how do we compute τ_c and τ^*?
- In the code we just saw, $\tau^* = \tau_K = \sqrt{\nu/\varepsilon}$
- 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.
- τ_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 τ_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 τ_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 τ_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 τ_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 τ_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: $\overline{\dot{\omega}_i} = \sum_{j=1}^{n_R} \gamma_j^* \overline{\dot{\omega}_{i,j,PSR}}$

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

```
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
- `reactions_`, the list of reactions
- `modRR_`, the locally modified production rates of all species

```
Y_(this->thermo().composition().Y()),
reactions_
(
    dynamic_cast<const reactingMixture<gasHThermoPhysics>&>(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

```
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 `ri`, 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.

```
tcSpecies =
    mag(rhoC*Y_[i][celli]/(refRRC*W*stoichCoeff));
```

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

```
if (tcSpecies < tc)
{
    tc = tcSpecies;
}
```

Implementing foxPaSR

The mixing time scale is also computed.

```
tk =
    sqrt(max(muC/rhoC/(epsC + SMALL), 0));

tp =
    max(tkeC/(epsC + SMALL), 0);

tm = sqrt(tk*tp);
```

$\tau_K = \sqrt{\nu/\varepsilon}$

$\tau_\Delta = \Delta/v'$

$\tau^* = \sqrt{\tau_k \tau_\Delta}$

Implementing foxPaSR

The reaction-specific volume fraction γ_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.

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

Implementing foxPaSR

The scaled production rates in reaction `ri` are then added to the total modified production rates.

```
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.

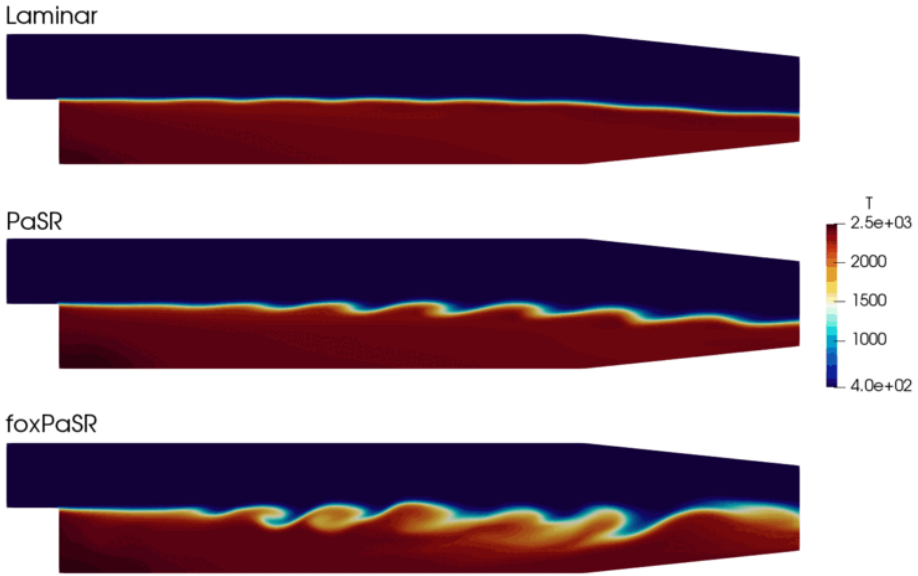
```
forAll(modRR_, i)
{
    this->chemistryPtr_->RR(i) = modRR_[i];
}
```

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

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



References

- Sabelnikov, V.; Fureby, C. LES Combustion Modeling for High Re Flames using a Multi-phase Analogy. Combust. Flame **2013**, 160, 83.
- Wartha, E.; Bösenhofer, M.; Harasek, M. Characteristic Chemical Time Scales for Reactive Flow Modeling. Combust. Sci. Technol. **2021**, 193, 2807.

Thanks for listening!

cfdfox.com