CEMA in OpenFOAM

Jaejung Kim

April 2025

1. Review of Chemical Kinetics

The chemical reaction rate w is a source term of the species conservation equation.

$$\frac{\partial \rho Y_i}{\partial t} + \nabla \cdot (\rho \vec{v} Y_i) = \nabla \cdot (\rho D \nabla Y_i) + w_i \qquad (1 \le k \le N)$$

Let's consider a forward reaction represented by

$$\sum_{i=1}^{N} \nu_i' M_i \rightarrow \sum_{i=1}^{N} \nu_i'' M_i$$

The rate of change in molar concentration c_i of species i is uniquely related to ω as

$$\hat{\omega}_i = \frac{dc_i}{dt} = (\nu_i'' - \nu_i')\omega$$

The phenomenological law of mass action states that ω is proportional to the product of the concentrations of the reactants.

$$\omega = k \prod_{i=1}^{N} c_i^{\nu_i'}$$

For an elementary reaction the Arrhenius law states that

$$k(T) = BT^{\alpha} e^{-E_a/R^o T}$$

Relation between mass fraction and concentration is given by

$$\rho Y_i \left[\frac{\text{kg}}{\text{m}^3}\right] = W_i c_i \left[\frac{\text{kg}}{\text{mol}} \cdot \frac{\text{mol}}{\text{m}^3}\right]$$

Therefore,

$$w_i = W_i \hat{\omega}_i$$

Finally, species reaction rate w_i is generalized for K forward reactions as

$$w_i = W_i \sum_{k=1}^{K} \left[(\nu_{i,k}'' - \nu_{i,k}') B_k T^{\alpha_k} e^{-E_{a,k}/R^o T} \prod_{j=1}^{N} c_j^{\nu_{j,k}'} \right]$$

2. Definition of Chemical Jacobian Matrix

Let's start from the system of

$$\rho \frac{dY_i}{dt} = w_i$$

Derivative of w is

$$\frac{dw_i}{dt} = \frac{\partial w_i}{\partial Y_j}\frac{dY_j}{dt} = \frac{\partial W_i \hat{\omega}_i}{\partial W_j c_j/\rho}\frac{w_j}{\rho} = \frac{\partial \hat{\omega}_i}{\partial c_j}\frac{W_i}{W_j} w_j$$

$$\frac{d\hat{\omega}_i}{dt} = \frac{\partial \hat{\omega}_i}{\partial c_j}\hat{\omega}_i$$

Chemical jacobian matrix is a linear operator of derivative.

$$J_{ij} = \frac{\partial \hat{\omega}_i}{\partial c_j} \qquad (1 \le i \le N,\ \ 1 \le j \le N)$$

Actually, chemical jacobian in combustion system is defined by additionally considering temperature.

$$J_{i,N+1} = \frac{\partial \hat{\omega}_i}{\partial T} \qquad (1 \le i \le N)$$

3. Concept of Chemical Explosive Mode Analysis (CEMA)

Diagonalization of J is given by

$$J = P^{-1}\Lambda P$$

Vector of modes f is defined by

$$f = P\hat{\omega}$$

so that

$$\frac{df}{dt} = \Lambda f$$

Since Λ is diagonal matrix of eigenvalues λ, analytic solution of the differential equation is given by

$$f_i = A_i e^{\lambda_i t}$$

Let λ_i is a complex value $a + bi$ then

$$f_i = A_i e^{at}[\cos(bt) + i\sin(bt)]$$

The real part represents the reciprocal time sale of the explosion, while the imaginary part is the oscillation frequency. But we focus only on the real part. Due to the conservation of elements and energy, J typically has $M + 1$ zero eigenvalues, where M is the number of elements involved in the species. This can be mathematically proved by Rank-nullity theorem.

Theorem 1. Rank-nullity theorem
Let $T : V \to W$ be a linear transformation between two vector spaces where T's domain V is finite dimensional. Then $\text{rank}(T) + \text{nullity}(T) = \dim V$, where $\text{rank}(T)$ is the rank of T (the dimension of its image) and $\text{nullity}(T)$ is the nullity of T (the dimension of its kernel).

λ_{exp} is defined by the eigenvalue with the largest real part among the non-conservative chemical modes to exclude the trivial zero eigenvalues.

$$\lambda_{\text{exp}} = \max(\mathfrak{Re}(\lambda \neq 0))$$

A chemical explosive mode is associated with an eigenvalue with a positive λ_{exp}, and nonexplosive mode is associated with an eigenvalue with a negative λ_{exp}.

4. Chemical Jacobian Matrix in OpenFOAM

Chemical jacobian matrix is generated in

```
chemistryModel/chemistryModel/StandardChemistryModel/StandardChemistryModel.C
...
template<class ReactionThermo, class ThermoType>
void Foam::StandardChemistryModel<ReactionThermo, ThermoType>::jacobian
...
    forAll(reactions_, ri)
    {
        const Reaction<ThermoType>& R = reactions_[ri];
        scalar kfwd, kbwd;
        R.dwdc(p, T, c_, J, dcdt, omegaI, kfwd, kbwd, false, dummy);
        R.dwdT(p, T, c_, omegaI, kfwd, kbwd, J, false, dummy, nSpecie_);
    }
```

dwdc and dwdT is obtained in

```
specie/reaction/Reactions/Reaction/Reaction.C
...
template<class ReactionThermo>
void Foam::Reaction<ReactionThermo>::dwdc
...
template<class ReactionThermo>
void Foam::Reaction<ReactionThermo>::dwdT
```

dwdc is theoretically derived as

$$\omega = k(T)\prod_{i=1}^{N} c_i^{\nu_i'}$$

$$\frac{d\omega}{dc_j} = k(T)\nu_j c_j^{\nu_j'-1}\prod_{i\neq j} c_i^{\nu_i'}$$

It is implemented as follow.

```
specie/reaction/Reactions/Reaction/Reaction.C
...
template<class ReactionThermo>
void Foam::Reaction<ReactionThermo>::dwdc
...
        scalar kf = kfwd;
        forAll(lhs_, i)
        {
            const label si = lhs_[i].index;
            const scalar el = lhs_[i].exponent;
            if (i == j)
            {
                if (el < 1)
                {
                    if (c[si] > SMALL)
                    {
                        kf *= el*pow(c[si] + VSMALL, el - 1);
                    }
                    else
                    {
                        kf = 0;
                    }
                }
                else
                {
                    kf *= el*pow(c[si], el - 1);
                }
            }
            else
            {
                kf *= pow(c[si], el);
            }
        }
```

dwdT is derived as

$$\omega = k(T)\prod_{i=1}^{N} c_i^{\nu_i'}$$

$$\frac{d\omega}{dT} = \frac{dk(T)}{dT}\prod_{i=1}^{N} c_i^{\nu_i'}$$

$$\frac{dk(T)}{dT} = \frac{d}{dT}AT^{\beta}e^{-T_a/T}$$

$$= AT^{\beta}e^{-T_a/T}\left(\frac{\beta}{T} + \frac{T_a}{T^2}\right)$$

It is implemented as follow.

```
specie/reaction/Reactions/Reaction/Reaction.C
...
template<class ReactionThermo>
void Foam::Reaction<ReactionThermo>::dwdT
...
    scalar kf = kfwd;
    scalar kr = kbwd;

    scalar dkfdT = this->dkfdT(p, T, c);
    scalar dkrdT = this->dkrdT(p, T, c, dkfdT, kr);

    scalar sumExp = 0.0;
    forAll(lhs_, i)
    {
        const label si = lhs_[i].index;
        const scalar el = lhs_[i].exponent;
        const scalar cExp = pow(c[si], el);
        dkfdT *= cExp;
        kf *= cExp;
        sumExp += el;
    }
    kf *= -sumExp/T;

    sumExp = 0.0;
    forAll(rhs_, i)
    {
        const label si = rhs_[i].index;
        const scalar er = rhs_[i].exponent;
        const scalar cExp = pow(c[si], er);
        dkrdT *= cExp;
        kr *= cExp;
        sumExp += er;
    }
    kr *= -sumExp/T;

    // dqidT includes the third-body (or pressure dependent) effect
    scalar dqidT = dkfdT - dkrdT + kf - kr;
```

```
specie/reaction/Reactions/ReversibleReaction/ReversibleReaction.C
...
Foam::scalar Foam::ReversibleReaction
<
    ReactionType,
    ReactionThermo,
    ReactionRate
>::dkfdT
(
    const scalar p,
    const scalar T,
    const scalarField& c
) const
{
    return k_.ddT(p, T, c);
}
```

```
specie/reaction/reactionRate/ArrheniusReactionRate/ArrheniusReactionRateI.H
...
inline Foam::scalar Foam::ArrheniusReactionRate::ddT
(
    const scalar p,
    const scalar T,
    const scalarField&
) const
{
    scalar ak = A_;

    if (mag(beta_) > vSmall)
    {
        ak *= pow(T, beta_);
    }

    if (mag(Ta_) > vSmall)
    {
        ak *= exp(-Ta_/T);
    }

    return ak*(beta_+Ta_/T)/T;
}
```

It can be noticed that two additional terms of kf and kr are included.

```
scalar dqidT = dkfdT - dkrdT + kf - kr;
```

They appear to be defined as

$$\mathrm{kf} = -\frac{\sum_{i=1}^{N} \nu_i'}{T} k_f(T) \prod_{i=1}^{N} c_i^{\nu_i'}$$

$$\mathrm{kr} = -\frac{\sum_{i=1}^{N} \nu_i''}{T} k_b(T) \prod_{i=1}^{N} c_i^{\nu_i''}$$

These terms appear to be of unclear origin and unnecessary. They are deleted and then

```
scalar dqidT = dkfdT - dkrdT;
```

This was a bug that lasted until the OpenFOAM-8, and was fixed in the OpenFOAM-9 version as below.

```
    scalar dkfdT = this->dkfdT(p, T, c, li);
    scalar dkrdT = this->dkrdT(p, T, c, li, dkfdT, kbwd);

    forAll(lhs(), i)
    {
        const label si = lhs()[i].index;
        const scalar el = lhs()[i].exponent;
        dkfdT *= c[si] >= small || el >= 1 ? pow(max(c[si], 0), el) : 0;
    }
    forAll(rhs(), i)
    {
        const label si = rhs()[i].index;
        const scalar er = rhs()[i].exponent;
        dkrdT *= c[si] >= small || er >= 1 ? pow(max(c[si], 0), er) : 0;
    }

    const scalar dqidT = dkfdT - dkrdT;
```

Implementation of CEMA in OpenFOAM

CEMAChemistryModel is copied from StandardChemistryModel.
Chemical jacobian matrix is copied, and a new function is added.

```
chemistryModel/chemistryModel/CEMAChemistryModel/CEMAChemistryModel.C
...
template<class ReactionThermo, class ThermoType>
void Foam::CEMAChemistryModel<ReactionThermo, ThermoType>::jacobian
...
    for (int i = 0; i < nSpecie_+1; ++i) {
        for (int j = 0; j < nSpecie_+1; ++j) {
            chemJacobian_(i, j) = J(i, j);
        }
    }
...
template<class ReactionThermo, class ThermoType>
void Foam::CEMAChemistryModel<ReactionThermo, ThermoType>::cema
(
    scalar& lambda
) const
{
    Eigen::MatrixXd J_eigen(nSpecie_+1,nSpecie_+1);
    for (int i = 0; i < nSpecie_+1; ++i) {
        for (int j = 0; j < nSpecie_+1; ++j) {
            J_eigen(i, j) = chemJacobian_(i, j);
        }
    }

    Eigen::EigenSolver<Eigen::MatrixXd> solver(J_eigen);
    Eigen::VectorXcd eigvals = solver.eigenvalues();
    std::vector<std::pair<double, int>> eigMagWithIndex;
    for (int i = 0; i < nSpecie_+1; ++i)
    {
        double magSq = std::norm(eigvals(i));
        eigMagWithIndex.emplace_back(magSq, i);
    }
    std::sort(eigMagWithIndex.begin(), eigMagWithIndex.end());
    std::vector<bool> isConserved(nSpecie_+1, false);

    for (int i = 0; i < nElements_+1; ++i)
    {
        int idx = eigMagWithIndex[i].second;
        isConserved[idx] = true;
    }

    lambda = -1e30;
    for (int i = 0; i < nSpecie_+1; ++i)
    {
        if (!isConserved[i])
        {
            double realPart = eigvals(i).real();
            if (realPart > lambda)
            {
                lambda = realPart;
            }
        }
    }
}
```

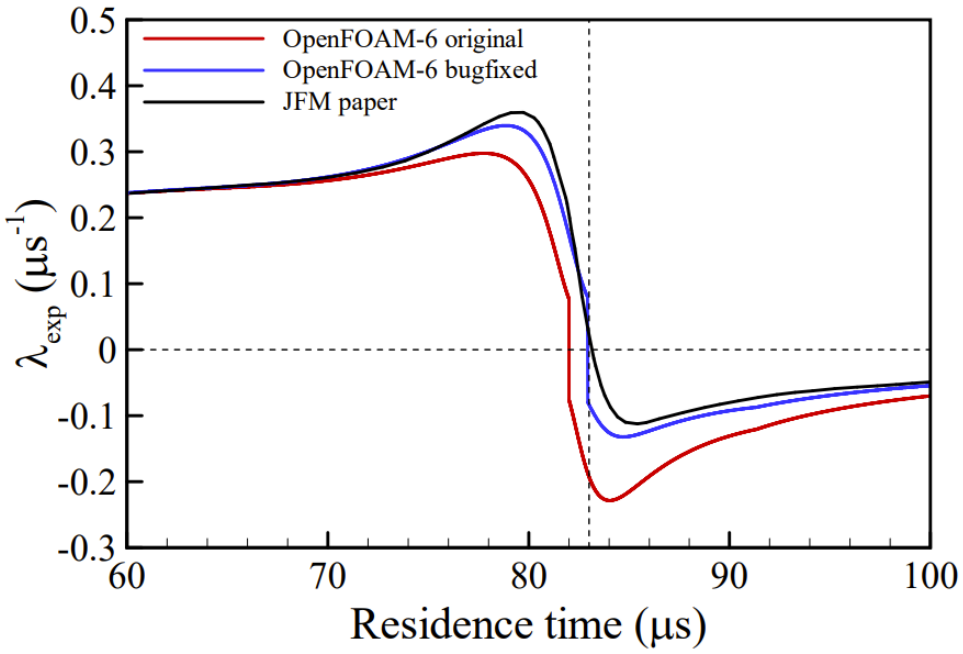

Figure 1: Effect of bug fix