Comparison of the governing equations found in the presentation of Wang et al., Status of FireFOAM Development and Future Plan, 2011, with the fireFoam source code included in OpenFOAM version 2.0.x build 2.0.x-931a91d59a3a

Convention:
Equations are made with MS equation editor
OpenFOAM source code is *black and cursive*
Comments to the source code are blue if understood, **red and bold** if not

Conservation of mass:

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_j}{\partial x_j} = 0$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/applications/solvers/combustion/fireFoam/rhoEqn.H

```
solve
  (
    fvm::ddt(rho)                 Change of mass with time
  + fvc::div(phi)                 Change of mass due to flow
  ==
    parcels.Srho(rho)             Source term for change due to parcels, e. g. from a Sprinkler
  + surfaceFilm.Srho()            Source term for change due to a surface film, e. g. due to water spray etc.
  );
```

Conservation of Momentum:

$$\frac{\partial \bar{\rho}\tilde{u}_i}{\partial t}+\frac{\partial \bar{\rho}\tilde{u}_i\tilde{u}_j}{\partial x_j}=\frac{\partial}{x_j}\left(\bar{\rho}(\nu+\nu_t)\left(\frac{\partial \tilde{u}_i}{\partial x_j}+\frac{\partial \tilde{u}_j}{\partial x_i}-\frac{2}{3}\frac{\partial \tilde{u}_k}{\partial x_k}\delta_{ij}\right)\right)-\frac{\partial \bar{p}}{\partial x_i}+\bar{\rho}g_i$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/applications/solvers/combustion/fireFoam/UEqn.H

```
fvVectorMatrix UEqn
(
    fvm::ddt(rho, U)                    Change of impulse with time
  + fvm::div(phi, U)                    Change of impulse due to flow
  + turbulence->divDevRhoReff(U)        Expression for turbulence, solved by OneEquationEddy model, see below?
 ==
    parcels.SU(U)
);

UEqn.relax();

if (pimple.momentumPredictor())         If this scheme is used, attach the two terms below; due to solver scheme reasons?
{
    solve
    (
        UEqn
     ==
        fvc::reconstruct
        (
            (
              - ghf*fvc::snGrad(rho)    Influence of pressure gradient on impulse
              - fvc::snGrad(p_rgh)      Influence of hydrostatic pressure on impulse
```

```
            )*mesh.magSf()
        )
    );
}
```

To solve (**I assume the expression below is not the one in the source code?)**

$$\frac{\partial}{x_j}\left(\overline{\rho}\left(\nu+\nu_t\right)\left(\frac{\partial\tilde{u}_i}{\partial x_j}+\frac{\partial\tilde{u}_j}{\partial x_i}-\frac{2}{3}\frac{\partial\tilde{u}_k}{\partial x_k}\delta_{ij}\right)\right)=\nabla\cdot\left(\overline{\rho}\tilde{\mathbf{u}}k\right)$$

the one equation eddy viscosity model is written as:

$$\frac{\partial\overline{\rho}k}{\partial t}+\nabla\cdot\left(\overline{\rho}\tilde{\mathbf{u}}k\right)=\nabla\cdot\left(\overline{\rho}\nu_k\nabla k\right)+P-\varepsilon$$

with assumed parts from /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/turbulenceModels/compressible/LES/oneEqEddy/oneEqEddy.C

$P=-\overline{\rho}\cdot(D:B)$ **Double inner product of two tensors, as noted in oneEqEddy.H?**

$\varepsilon=c_\varepsilon\overline{\rho}\sqrt[3]{k}\Delta^{-1}$

$D=\mathrm{symm}(\mathrm{grad}(U))$ **Symmetric part of a rank 2 tensor created by the outer product of gradient and velocity vector, as noted in oneEqEddy.H?**

$B=\frac{2}{3}kI-2\nu_{\mathrm{SGS}}\cdot\mathrm{dev}(D)$ **Sub grid stress tensor, what is I?**

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/turbulenceModels/compressible/LES/oneEqEddy/oneEqEddy.C

```
tmp<fvScalarMatrix> kEqn
(
    fvm::ddt(rho(), k_)
```

```
    + fvm::div(phi(), k_)
    - fvm::laplacian(DkEff(), k_)
   ==
     G
    - fvm::SuSp(2.0/3.0*rho()*divU, k_)
    - fvm::Sp(ce_*rho()*sqrt(k_)/delta(), k_)
  );
```

with G to be found in line 108 as:

*volScalarField G(2*muSgs_*(gradU && dev(symm(gradU))));*

leading to the full equation of:

$$\frac{\partial \bar{\rho} k}{\partial t} + \nabla \cdot \left(\bar{\rho} \tilde{\mathbf{u}} k \right) - \nabla \cdot \left(D_{k,\mathrm{eff}} \nabla k \right) = 2 \mu_{\mathrm{SGS}} \left(\nabla \tilde{\mathbf{u}} : \mathrm{dev}(D) \right) - \frac{2}{3} \bar{\rho} \nabla \cdot \left(\tilde{\mathbf{u}} k \right) - c_{\varepsilon} \bar{\rho} \sqrt{k} k \Delta^{-1}$$

Is this correct? Is this equation the same as the one in the presentation due to reformulation?

See also Yeoh and Yuen, Computational Fluid Dynamics in Fire Engineering, 1. ed., 2009, eq. 5.2.69, p. 388.

Conservation of sensible enthalpy:

$$\frac{\partial \bar{\rho}\tilde{h}_s}{\partial t}+\frac{\partial \bar{\rho}\tilde{u}_j\tilde{h}_s}{\partial x_j}=\frac{D\bar{p}}{Dt}+\frac{\partial}{x_j}\left(\bar{\rho}\left(D_{\text{th}}+\frac{\nu_t}{\text{Pr}_t}\right)\frac{\partial \tilde{h}_s}{\partial x_j}\right)-\frac{\partial \bar{\dot{q}}''}{\partial x}+\bar{\dot{\omega}}_{h_s}'''$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/applications/solvers/combustion/fireFoam/YhsEqn.H

```
fvScalarMatrix hsEqn
  (
    fvm::ddt(rho, hs)                                  Change of enthalpy with time
   + mvConvection->fvmDiv(phi, hs)                     Change of enthalpy due to convection
   - fvm::laplacian(turbulence->alphaEff(), hs)        Change of enthalpy due to diffusion, see below
  ==
    DpDt                                               Influence of pressure change work?
   + dQ                                                Influence of heat release rate
   + radiation->Shs(thermo)                            Change due to heat radiation, see fvDOM model below
   + parcels.Sh(hs)                                    Source term for change due to parcels, e. g. from a Sprinkler
   + surfaceFilm.Sh()                                  Source term for change due to a surface film, e. g. due to water spray etc.
  );
```

If one rewrites the diffusion term to

$$\frac{\partial}{x_j}\left(\bar{\rho}\left(D_{\text{th}}+\frac{\nu_t}{\text{Pr}_t}\right)\frac{\partial \tilde{h}_s}{\partial x_j}\right)=\bar{\rho}D_{\text{th}}\frac{\partial^2 \tilde{h}_s}{\partial x_j^{\,2}}+\bar{\rho}\frac{\nu_t}{\text{Pr}_t}\frac{\partial^2 \tilde{h}_s}{\partial x_j^{\,2}}$$

The former term denotes the molecular diffusion, the ladder the turbulent diffusion?

Conservation of species mass fraction

$$\frac{\partial \bar{\rho}\tilde{Y}_k}{\partial t}+\frac{\partial \bar{\rho}\tilde{u}_j\tilde{Y}_k}{\partial x_j}=\frac{\partial}{x_j}\left(\bar{\rho}\left(D_k+\frac{\nu_t}{\mathrm{Pr}_t}\right)\frac{\partial \tilde{Y}_k}{\partial x_j}\right)-+\bar{\dot{\omega}}_{Y_k}^{'''}$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/applications/solvers/combustion/fireFoam/YhsEqn.H

```
fvScalarMatrix YiEqn
    (
      fvm::ddt(rho, Yi)                          Change of species mass fraction with time
     + mvConvection->fvmDiv(phi, Yi)             Change of species mass fraction due to convection
     - fvm::laplacian(turbulence->alphaEff(), Yi)   Change of species mass fraction due to diffusion, see below
     ==
      parcels.SYi(i, Yi)                         Source term for change due to parcels, e. g. from a Sprinkler
     + surfaceFilm.Srho(i)                       Source term for change due to a surface film, e. g. due to water spray etc.
     + R                                         Other source terms, e. g. from combustion (fuel consumption matrix)?
    );
```

If one rewrites the diffusion term to

$$\frac{\partial}{x_j}\left(\bar{\rho}\left(D_k+\frac{\nu_t}{\mathrm{Pr}_t}\right)\frac{\partial \tilde{Y}_k}{\partial x_j}\right)=\bar{\rho}D_{\mathrm{k}}\frac{\partial^2\tilde{Y}_k}{\partial x_j^{\,2}}+\bar{\rho}\frac{\nu_t}{\mathrm{Pr}_t}\frac{\partial^2\tilde{Y}_k}{\partial x_j^{\,2}}$$

The former term denotes the molecular species diffusion, the ladder the turbulent species diffusion?

Other equations identified in the source code:

Equation of state, **from ?**

$$\bar{p} = \bar{\rho} R \tilde{T}$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/turbulenceModels/compressible/LES/oneEqEddy/oneEqEddy.C, l. 47 and 50

$$\alpha_{\mathrm{SGS}} = \frac{\mu_{\mathrm{SGS}}}{\mathrm{Pr}_{\mathrm{t}}} \quad \mathrm{Pr}_{\mathrm{t}} = \frac{\mu_{\mathrm{SGS}}}{\alpha_{\mathrm{SGS}}}$$

$$\mu_{\mathrm{SGS}} = c_k \bar{\rho} \sqrt{k} \Delta$$

It follows from $\mu = \nu \rho$

$$\nu_{\mathrm{SGS}} \bar{\rho} = c_k \bar{\rho} \sqrt{k} \Delta$$

and **assuming (?)** $\nu_t = \nu_{\mathrm{SGS}}$

$$\nu_{\mathrm{t}} = c_k \sqrt{k} \Delta$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/turbulenceModels/compressible/LES/oneEqEddy/oneEqEddy.H, l. 130

$$D_{k,\mathrm{eff}} = \mu_{\mathrm{laminar}} + \mu_{\mathrm{SGS}}$$

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/turbulenceModels/compressible/LES/LESModel/LESModel.H, l. 202 and 214

$$\mu_{\mathrm{eff}} = \mu_{\mathrm{SGS}} + \mu$$

$$\alpha_{\mathrm{eff}} = \alpha_{\mathrm{SGS}} + \alpha$$

The return of alpha is different for a patch!

From /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/turbulenceModels/LES/LESdeltas/cubeRootVolDelta/cubeRootVolDelta.C, l. 48

$$\Delta = c_{\Delta} \frac{1}{3} V$$

Combustion

Default combustion model is with infinitelyFastChemistry, working with singleReactionMixture, as noted in the tutorials "combustionproperties" files.

In /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/thermophysicalModels/reactionThermo/mixtures/singleStepReactingMixture/singleStepReactingMixture.C

```
void Foam::singleStepReactingMixture<ThermoType>::calculateqFuel()
{
    const Reaction<ThermoType>& reaction = this->operator[](0);
    const  scalar Wu = this->speciesData()[fuelIndex_].W();

    forAll(reaction.lhs(), i)                          Calculation of the reactants side of reaction equation
    {
        const label specieI = reaction.lhs()[i].index;
        const scalar stoichCoeff = reaction.lhs()[i].stoichCoeff;
        specieStoichCoeffs_[specieI] = -stoichCoeff;
        qFuel_.value() += this->speciesData()[specieI].hc()*stoichCoeff/Wu;
    }

    forAll(reaction.rhs(), i)                          Calculation of the products side of reaction equation
    {
        const label specieI = reaction.rhs()[i].index;
        const scalar stoichCoeff = reaction.rhs()[i].stoichCoeff;
        specieStoichCoeffs_[specieI] = stoichCoeff;
        qFuel_.value() -= this->speciesData()[specieI].hc()*stoichCoeff/Wu;
        speciesProd_[specieI] = -1;
    }

    Info << "Fuel heat of combustion :" << qFuel_.value() << endl;
```

```
}
```

As an equation for reactants species:

$$\Delta h_{c,i} = h_{c,i} \frac{-\nu_i}{W_f}$$

and for product species:

$$\Delta h_{c,i} = h_{c,i} \frac{\nu_i}{W_f}$$

The total fuel heat of combustion is expressed as:

$$\Delta h = \sum_{i=1}^{n} \Delta h_{c,i}$$

Next it follows the computation of the stoichiometric air-fuel and oxygen-fuel ratio:

```
void Foam::singleStepReactingMixture<ThermoType>::massAndAirStoichRatios()
{
    const label O2Index = this->species()["O2"];
    const scalar Wu = this->speciesData()[fuelIndex_].W();

    stoicRatio_ =                                          Stoichiometric air-fuel ratio
        (this->speciesData()[inertIndex_].W()
      * specieStoichCoeffs_[inertIndex_]
      + this->speciesData()[O2Index].W()
      * mag(specieStoichCoeffs_[O2Index]))
      / (Wu*mag(specieStoichCoeffs_[fuelIndex_]));
```

```
    s_ =                                                    Stoichiometric oxygen-fuel ratio
        (this->speciesData()[O2Index].W()
      * mag(specieStoichCoeffs_[O2Index]))
      / (Wu*mag(specieStoichCoeffs_[fuelIndex_]));

    Info << "stoichiometric air-fuel ratio :" << stoicRatio_.value() << endl;

    Info << "stoichiometric oxygen-fuel ratio :" << s_.value() << endl;
}
```

Expressed mathematically if nitrogen is the inert, oxygen the oxidizing specie:

$$s_{\text{air,f}} = W_{\text{N}_2}\nu_{\text{N}_2} + \frac{W_{\text{O}_2}\nu_{\text{O}_2}}{W_{\text{f}}\nu_{\text{f}}}$$

$$s_{\text{O}_2,\text{f}} = \frac{W_{\text{O}_2}\nu_{\text{O}_2}}{W_{\text{f}}\nu_{\text{f}}}$$

Not completley clear section:

```
void Foam::singleStepReactingMixture<ThermoType>::calculateMaxProducts()
{
    const Reaction<ThermoType>& reaction = this->operator[](0);

    scalar Wm = 0.0;
    scalar totalMol = 0.0;
    forAll(reaction.rhs(), i)                    Total number of moles on the product side of the reaction
    {
        label specieI = reaction.rhs()[i].index;
```

```
        totalMol += mag(specieStoichCoeffs_[specieI]);
    }

    scalarList Xi(reaction.rhs().size());

    forAll(reaction.rhs(), i)
    {
        const label specieI = reaction.rhs()[i].index;
        Xi[i] = mag(specieStoichCoeffs_[specieI])/totalMol;

        Wm += Xi[i]*this->speciesData()[specieI].W();
    }

    forAll(reaction.rhs(), i)
    {
        const label specieI = reaction.rhs()[i].index;
        Yprod0_[specieI] =  this->speciesData()[specieI].W()/Wm*Xi[i];
    }

    // Normalize the stoichiometric coeff to mass
    forAll(specieStoichCoeffs_, i)
    {
        specieStoichCoeffs_[i] =
            specieStoichCoeffs_[i]
          * this->speciesData()[i].W()
          / (this->speciesData()[fuelIndex_].W()
          * mag(specieStoichCoeffs_[fuelIndex_]));
    }
}
```

Mole fractions of generated products?

Total molecular mass?

What shall Yprod,0 be? Mass fractions at time t=0?

Why has this to be done?

Math:

Total number of product moles with p products:

$$n = \sum_{i=1}^{p} n_{p,i}$$

Mole fraction (?):

$$X_i = \frac{n_{p,i}}{n}$$

Total molecular mass?

$$W_m = \sum_{i=1}^{p} X_i W_i$$

Mass fraction of product?

$$Y_{\text{prod},0} = \frac{W_i}{W_m X_i}$$

Mass normalized stoichiometric coefficents:

$$\nu_{m,i} = \frac{\nu_i W_i}{\nu_f W_f}$$

Followd by the calculation of "fres" for every species, different for fuel, oxygen and products. It is understood, that this is per species and cell, but I could'nt figure out what it is.

void Foam::singleStepReactingMixture<ThermoType>::fresCorrect()
{

```
const Reaction<ThermoType>& reaction = this->operator[](0);

label O2Index = this->species()["O2"];
const volScalarField& YFuel = this->Y()[fuelIndex_];
const volScalarField& YO2 = this->Y()[O2Index];

// reactants
forAll(reaction.lhs(), i)
{
    const label specieI = reaction.lhs()[i].index;
    if (specieI == fuelIndex_)
    {
        fres_[specieI] =  max(YFuel - YO2/s_, scalar(0.0));
    }
    else if (specieI == O2Index)
    {
        fres_[specieI] =  max(YO2 - YFuel*s_, scalar(0.0));
    }
}


// products
forAll(reaction.rhs(), i)
{
    const label specieI = reaction.rhs()[i].index;
    if (specieI != inertIndex_)
    {
        forAll(fres_[specieI], cellI)
        {
            if (fres_[fuelIndex_][cellI] > 0.0)
            {
                // rich mixture
```

What is a "fres"?

What is a "fres"?

```
                fres_[specieI][cellI] =
                    Yprod0_[specieI]
                  * (1.0 + YO2[cellI]/s_.value() - YFuel[cellI]);
            }
            else
            {
                // lean mixture
                fres_[specieI][cellI] =
                    Yprod0_[specieI]
                  * (
                        1.0
                      - YO2[cellI]/s_.value()*stoicRatio_.value()
                      + YFuel[cellI]*stoicRatio_.value()
                    );
            }
        }
    }
  }
}
```

Using the obtained values with /Volumes/Mac-OF-2.0.x/OpenFOAM-2.0.x/src/combustionModels/infinitelyFastChemistry/infinitelyFastChemistry.C:

```
void Foam::combustionModels::infinitelyFastChemistry::correct()
{
    singleMixture_.fresCorrect();

    const label fuelI = singleMixture_.fuelIndex();

    const volScalarField& YFuel = thermo_.composition().Y()[fuelI];
```

```
    const dimensionedScalar s = singleMixture_.s();

    if (thermo_.composition().contains("O2"))
    {
        const volScalarField& YO2 = thermo_.composition().Y("O2");
        wFuelNorm_ == rho_/(mesh_.time().deltaT()*C_)*min(YFuel, YO2/s.value());

    }
}
```

Calculation of the normalized consumption rate of fuel ?

What is (mesh_.time().deltaT()*C_) here?

```
Foam::tmp<Foam::fvScalarMatrix>
Foam::combustionModels::infinitelyFastChemistry::R(volScalarField& Y) const
{
    const label specieI = thermo_.composition().species()[Y.name()];

    const label fNorm = singleMixture_.specieProd()[specieI];

    const volScalarField fres(singleMixture_.fres(specieI));

    const volScalarField wSpecie
    (
        wFuelNorm_*singleMixture_.specieStoichCoeffs()[specieI]
      / max(fNorm*(Y - fres), scalar(0.001))
    );

    return -fNorm*wSpecie*fres + fNorm*fvm::Sp(wSpecie, Y);
}
```

This is what?

Calculation of species production/consumption rate?

```
Foam::tmp<Foam::volScalarField>
Foam::combustionModels::infinitelyFastChemistry::dQ() const
{
    const label fuelI = singleMixture_.fuelIndex();
    volScalarField& YFuel = thermo_.composition().Y(fuelI);

    return -singleMixture_.qFuel()*(R(YFuel) & YFuel);
}
```

Calculation of the heat release rate?