How to implement your own turbulence model (1/3)

- The implementations of the turbulence models are located in `$FOAM_SRC/turbulenceModels`
- Copy the source of the turbulence model that is most similar to what you want to do. In this case we will make our own copy of the `kEpsilon` turbulence model:

```
cd $WM_PROJECT_DIR
cp -r --parents src/turbulenceModels/RAS/incompressible/kEpsilon \
      $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/turbulenceModels/RAS/incompressible/
mv kEpsilon mykEpsilon
cd mykEpsilon
```

How to implement your own turbulence model (2/3)

- We also need a Make/files and a Make/options
- Use the original files as a base:
  `$FOAM_SRC/turbulenceModels/RAS/incompressible/Make`
- `Make/files`:

```
mykEpsilon.C
LIB = $(FOAM_USER_LIBBIN)/libmyIncompressibleRASModels
```

  (we are only *adding* `mykEpsilon`)

- `Make/options`:

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/turbulenceModels/RAS/incompressible/lnInclude
LIB_LIBS =
```

  (the last `-I` is needed since `mykEpsilon` uses include-files in the original directory)

How to implement your own turbulence model (3/3)

- We need to modify the file names of our new turbulence model:

```
rename 's/kEpsilon/mykEpsilon/' *
```

- In `mykEpsilon.C` and `mykEpsilon.H`, change all occurances of `kEpsilon` to `mykEpsilon` so that we have a new class name:

```
sed s/kEpsilon/mykEpsilon/g mykEpsilon.C > temp
mv temp mykEpsilon.C
sed s/kEpsilon/mykEpsilon/g mykEpsilon.H > temp
mv temp mykEpsilon.H+
```

- Introduce a small modification so that we can see if we use our new model. Line 122 in mykEpsilon.C:

```
Info << "Defining my own kEpsilon model" << endl;
```

- Compile using:

```
wmake libso
```

which will build a dynamic library.

How to use your own turbulence model

- Tell OpenFOAM to use your new library by adding a line to `controlDict`:
  `libs ("libmyIncompressibleRASModels.so");`
- You choose turbulence model in the `constant/RASProperties` dictionary:

```
RASModel mykEpsilon;
```

- You also have to set the needed coefficients for your turbulence model, which in this case will be the same as for the `kEpsilon` model:

```
mykEpsilonCoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphaEps        0.76923;
}
```

- Now you can run the `simpleFoam/pitzDaily` tutorial with your new turbulence model. Try both `kEpsilon` and `mykEpsilon` and look for your write-statement in the log file.
- Simply add appropriate source terms to implement a variant of kEpsilon.

A note on new turbulence models

- The `RAS` turbulence models in OpenFOAM are sub-classes to the virtual class `RASModel`.
- You are only allowed to use the same member function definitions as in the `RASModel` class. If you need other member functions you will have to add those to the `RASModel` class, which requires that you copy and modify all of `$FOAM_SRC/turbulenceModels/RAS/incompressible`.
  You can recognize where the top-level of a class is located by locating the `Make`-directory.

We will now have a look at the implementation of the kOmegaSST model.

$k-\omega$ SST in OpenFOAM-1.5

From `$FOAM_SRC/turbulenceModels/RAS/incompressible/kOmegaSST/kOmegaSST.H`:

```
- Menter, F., Esch, T.
  "Elements of Industrial Heat Transfer Prediction"
  16th Brazilian Congress of Mechanical Engineering (COBEM),
  Nov. 2001
- Note that this implementation is written in terms of alpha diffusion
  coefficients rather than the more traditional sigma (alpha = 1/sigma)
  so that the blending can be applied to all coefficients in a
  consistent manner. The paper suggests that sigma is blended but this
  would not be consistent with the blending of the k-epsilon and
  k-omega models.
- Also note that the error in the last term of equation (2) relating to
  sigma has been corrected.
- Wall-functions are applied in this implementation by using equations
  (14) to specify the near-wall omega as appropriate.
- The blending functions (15) and (16) are not currently used because of
  the uncertainty in their origin, range of applicability and that is y+
  becomes sufficiently small blending u_tau in this manner clearly
  becomes nonsense.
```

$k-\omega$ SST in OpenFOAM-1.5, ν_t

Source code:

`$FOAM_SRC/turbulenceModels/RAS/incompressible/kOmegaSST`

Kinematic eddy viscosity:

$$\nu_t = \frac{a_1 k}{\max(a_1\omega, SF_2)}$$

Line 395, kOmegaSST.C:

```
nut_ = a1_*k_/max(a1_*omega_, F2()*sqrt(S2));
```

In `<case>/constant/RASProperties`, `kOmegaSSTCoeffs`:

```
a1                  0.31;
```

Line 343, kOmegaSST.C (S = `sqrt(S2)`):

```
volScalarField S2 = magSqr(symm(fvc::grad(U_)));
```

i.e. $S^2 = \left|\frac{1}{2}(\partial_j u_i + \partial_i u_j)\right|^2$ and $S = \sqrt{S^2} = \left|\frac{1}{2}(\partial_j u_i + \partial_i u_j)\right|$

F2() is a blending function, which is described on the next slide

$k-\omega$ SST in OpenFOAM-1.5, F2()

F2() is a blending function:

$$F_2 = \tanh\left[\left[\min\left(\max\left(\frac{2\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right), 100\right)\right]^2\right]$$

Lines 72-85, kOmegaSST.C:

```
tmp<volScalarField> kOmegaSST::F2() const
{
    volScalarField arg2 = min
    (
        max
        (
            (scalar(2)/betaStar_)*sqrt(k_)/(omega_*y_),
            scalar(500)*nu()/(sqr(y_)*omega_)
        ),
        scalar(100)
    );

    return tanh(sqr(arg2));
}
```

$k-\omega$ SST in OpenFOAM-1.5, Turbulence kinetic energy eq.

$$\frac{\partial k}{\partial t}+U_j\frac{\partial k}{\partial x_j}=P_k-\beta^* k\omega+\frac{\partial}{\partial x_j}\left[(\nu+\sigma_k\nu_T)\frac{\partial k}{\partial x_j}\right],\quad P_k=\min(G,10\beta^* k\omega),\quad G=\nu_t\frac{\partial U_i}{\partial x_j}\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right)$$

Lines 378-387, kOmegaSST.C:

```
tmp<fvScalarMatrix> kEqn
(
    fvm::ddt(k_)
  + fvm::div(phi_, k_)
  - fvm::Sp(fvc::div(phi_), k_)
  - fvm::laplacian(DkEff(F1), k_)
 ==
    min(G, c1_*betaStar_*k_*omega_)
  - fvm::Sp(betaStar_*omega_, k_)
);
```

The effective diffusivity for k, (`DkEff(F1)`), is described on a later slide.
F1 is obtained from F1(), which is a blending function for σ_k, and is described on the next slide,

where $CD_{k\omega}=2\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}$

```
volScalarField CDkOmega =
    (2*alphaOmega2_)*(fvc::grad(k_) & fvc::grad(omega_))/omega_;
```

F1() is a blending function, lines 47-70, kOmegaSST.C (compressed here):

$$F_1 = \tanh\left\{\left\{\min\left(\min\left[\max\left(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\sigma_{\omega 2}k}{CD^+_{k\omega}y^2}\right], 10\right)\right\}^4\right\}$$

```
tmp<volScalarField> kOmegaSST::F1(const volScalarField& CDkOmega) const
{   volScalarField CDkOmegaPlus = max
    (   CDkOmega,
        dimensionedScalar("1.0e-10", dimless/sqr(dimTime), 1.0e-10)
    );
    volScalarField arg1 = min
    ( min
        ( max
            (  (scalar(1)/betaStar_)*sqrt(k_)/(omega_*y_),
               scalar(500)*nu()/(sqr(y_)*omega_)
            ),
            (4*alphaOmega2_)*k_/(CDkOmegaPlus*sqr(y_))
        ),
        scalar(10)
    );
    return tanh(pow4(arg1));}
```

$F_1 = 0$ in the freestream ($k - \varepsilon$ model) and $F_1 = 1$ in the boundary layer ($k - \omega model$)

$k-\omega$ SST in OpenFOAM-1.5, Effective diffusivity for k

The effective diffusivity for k, (`DkEff(F1)`), lines 218-224, kOmegaSST.H:

```
tmp<volScalarField> DkEff(const volScalarField& F1) const
{
    return tmp<volScalarField>
    (
        new volScalarField("DkEff", alphaK(F1)*nut_ + nu())
    );
}
```

Blend `alphaK1` and `alphaK2` using blend function `F1`, lines 127-133, kOmegaSST.H:

```
tmp<volScalarField> alphaK
(
    const volScalarField& F1
) const
{
    return blend(F1, alphaK1_, alphaK2_);
}
```

In `<case>/constant/RASProperties`, `kOmegaSSTCoeffs`:

```
    alphaK1         0.85034;
    alphaK2         1.0;
```

$k-\omega$ SST in OpenFOAM-1.5, Specific dissipation rate eq.

$$\frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} = \alpha S^2 - \beta \omega^2 + \frac{\partial}{\partial x_j}\left[(\nu + \sigma_\omega \nu_T)\frac{\partial \omega}{\partial x_j}\right] + 2(1-F_1)\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}$$

Lines 354-368, kOmegaSST.C:

```
tmp<fvScalarMatrix> omegaEqn
(
    fvm::ddt(omega_)
  + fvm::div(phi_, omega_)
  - fvm::Sp(fvc::div(phi_), omega_)
  - fvm::laplacian(DomegaEff(F1), omega_)
 ==
    gamma(F1)*2*S2
  - fvm::Sp(beta(F1)*omega_, omega_)
  - fvm::SuSp
    (
        (F1 - scalar(1))*CDkOmega/omega_,
        omega_
    )
);
```