The implementation of Two-equation SGS turbulence model and Wall damping function

Yeru Shang

Thermo-fluid Mechanics Research Centre,
The University of Sussex,
Brighton, UK

2017-11-21

Acknowledgements

- Course Lecturer: Prof. Håkan Nilsson
- Technical Support: Mohammad Hossein Arabnejad Khanouki
- Supervisors: Dr. Esra Sorguven and Prof. Martin Rose

Content

- Technical background
- Model implementation
- Test case setup and post-processing
- Conclusion and further work

Filtered/averaged Navier-Stokes equations

Familiar with these?

$$\nabla \cdot \bar{\boldsymbol{U}} = 0$$

$$\frac{\partial \bar{\boldsymbol{U}}}{\partial t} + \nabla \cdot (\bar{\boldsymbol{U}}\bar{\boldsymbol{U}}) = -\nabla \bar{p} + \nabla \cdot \nu(\nabla \bar{\boldsymbol{U}} + \nabla \bar{\boldsymbol{U}}^T) - \nabla \cdot \tau$$

Current way of solving it: RANS, URANS, DES, Hybrid-LES, LES...
May be many years later, DNS.

Eddy Viscosity Theory

Newtons law of viscosity for incompressible flow

$$\tau_{Newtonian} = 2\nu S = \nu(\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T)$$

Also Turbulent stresses $\propto$ Mean rate of deformation
Boussinesq hypothesis for RANS:

$$\tau = -2\nu_t \overline{S} + \frac{1}{3}k\boldsymbol{I} = -\nu_t(\nabla \overline{\boldsymbol{U}} + \nabla \overline{\boldsymbol{U}}^T) + \frac{2}{3}k\boldsymbol{I}$$

Same theory can be adopted on LES.

On dimensional grounds:

$$\nu_t = C\vartheta\ell$$

Turbulence model based on Eddy Viscosity Theory is to find appropriate representation of ϑ and ℓ in order to model ν_t, thereby closing the filtered/averaged NSEs.

Most (if not all) of the popular models are eddy viscosity turbulence model, e.g. k-ε series, k-ω series, etc.

Inagi Wall Damping Function

Original k-equation SGS model evaluate eddy viscosity via

$$\nu_t = C_k \Delta \sqrt{k_{sgs}}$$

Inagi et al. proposed a Wall damping function for SGS model containing k-equation

$$\nu_t = F_{wY} C_k \Delta \sqrt{k_{sgs}}$$

where

$$F_{wY} = \frac{1}{1 + \Delta \sqrt{2|S_{ij}|^2} / C_T \sqrt{k_{sgs}}}$$

and $C_T = 10.0$.

Two-equation SGS model

The new Two-equation SGS model to be implemented is

$$\tau = -\alpha \nu_t (\nabla \overline{\boldsymbol{U}} + \nabla \overline{\boldsymbol{U}}^T) + \frac{2}{3} k_{sgs} \boldsymbol{I}$$

$$\nu_t = C_\mu \frac{k_{sgs}^2}{\varepsilon_{sgs}} \left(\frac{k_{sgs}}{k_{sgs} + k_r} \right)$$

$$\frac{\partial k_{sgs}}{\partial t} + \nabla \cdot (k_{sgs} \bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k_{sgs} \right] + \alpha P - \varepsilon_{sgs}$$

$$\frac{\partial \varepsilon_{sgs}}{\partial t} + \nabla \cdot (\varepsilon_{sgs} \bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \nabla \varepsilon_{sgs} \right] + \frac{\varepsilon_{sgs}}{k_{sgs}} \left[C_{\varepsilon 1} P - C_{\varepsilon 2} \varepsilon_{sgs} \right]$$

$$P = \nu_t(\nabla\overline{\boldsymbol{U}} + \nabla\overline{\boldsymbol{U}}^T)\nabla\overline{\boldsymbol{U}}; \qquad C_{\varepsilon 2} = (11/6)f + (25/Re_T)f^2$$

$$Re_T = k_{sgs}^2/\nu\varepsilon_{sgs}; \quad f = (Re_T/30[\sqrt{1+60/Re_T} - 1]$$

$$\alpha = 1.5\left\{1 - C^*\left(\frac{k_{sgs}}{k_{sgs}+k_r}\right)^2\left[\left(\frac{\Delta x_i}{\sqrt{k_r}}\frac{\partial\sqrt{k_r}}{\partial x_i}\right)^2 + 0.11\right]^{-1}\right\}$$

$$\frac{\left(\Delta x_i\frac{\partial\sqrt{k_r}}{\partial x_i}\right)^2}{k_r} = \frac{\left\{\left(\Delta x\frac{\partial\sqrt{k_r}}{\partial x}\right)^2 + \left(\Delta y\frac{\partial\sqrt{k_r}}{\partial y}\right)^2 + \left(\Delta z\frac{\partial\sqrt{k_r}}{\partial z}\right)^2\right\}}{k_r}$$

$$C_\mu = 0.18; \quad C_{\varepsilon 1} = 1.55; \quad \sigma_k = 1.0; \quad \sigma_\varepsilon = 1.2; \quad C^* = 0.28$$

Turbulence Model Library

The templated turbulence model class locates at:
`$FOAM_SRC/TurbulenceModels`

The available SGS model can found via:
`ls $FOAM_SRC/TurbulenceModels/turbulenceModels/LES`

The available RANS model can found via:
`ls $FOAM_SRC/TurbulenceModels/turbulenceModels/RAS`

The existing models which are close to the one being implemented is `kEqn` SGS model and `kEpsilon` RANS model. Let's have a look...

Standard k-ε RANS Model

$$\nu_t = C_\mu \frac{\sqrt{k}}{\varepsilon}$$

$$\frac{\partial k}{\partial t} + \nabla \cdot (k\bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k \right] + P - \varepsilon$$

$$\frac{\partial \varepsilon}{\partial t} + \nabla \cdot (\varepsilon\bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \nabla \varepsilon \right] + \frac{\varepsilon}{k} \left[C_{\varepsilon 1} P - C_{\varepsilon 2} \varepsilon \right]$$

$$C_\mu = 0.09; \quad C_{\varepsilon 1} = 1.44; \quad C_{\varepsilon 2} = 1.92; \quad \sigma_k = 1.0; \quad \sigma_\varepsilon = 1.3$$

Standard k-equation SGS Model

$$\nu_t = C_k \Delta \sqrt{k_{sgs}}$$

$$\frac{\partial k_{sgs}}{\partial t} + \nabla \cdot (k_{sgs}\bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\frac{\nu + \nu_t}{\sigma_k} \right) \nabla k_{sgs} \right] + P - \varepsilon_{sgs}$$

$$C_k = 0.094; \quad C_\varepsilon = 1.048; \quad \sigma_k = 1.0;$$

Let's have a look of their source code make a comparison:

```
cd $FOAM_SRC/TurbulenceModels
vi turbulenceModels/RAS/kEpsilon/kEpsilon.C
vi turbulenceModels/LES/kEqn/kEqn.C
```

A Tour To Turbulence Model Library

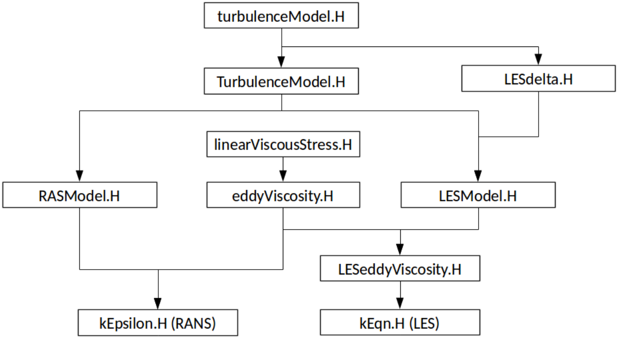

Figure: The Eco-system of the two turbulence models.

Let's do some dissections!

- `turbulenceModel.H`: Member Data and Member Functions;
- `TurbulenceModel.H`: line 126 for `Selectors`;
- `RASModel.H` vs `LESModel.H`;
- `linearViscousStress.H`: line 91 for `divDevRhoReff()`;

$$\nabla \cdot \tau = \nabla \cdot \left[-\frac{2}{3}\nu_e \nabla \cdot \boldsymbol{U}\boldsymbol{I} + \nu_e \nabla \boldsymbol{U} + \nu_e (\nabla \boldsymbol{U})^T \right] = \nabla \cdot (\nu_e \nabla \boldsymbol{U}) + \nabla \cdot \left(\nu_e \left[(\nabla \boldsymbol{U})^T + \frac{1}{3}\mathrm{tr}(\nabla \boldsymbol{U})^T \boldsymbol{I} \right] \right)$$

- `eddyViscosity.H`: line 105 for `nut()` and line 121 for `R()`;
- `LESeddyViscosity.H`: line71 for `Ce_` and line 108 for `epsilon()`;
- `LESdelta.H`: line 104 for `Selectors`.

Recall the Two-equation SGS model

The new Two-equation SGS model to be implemented is

$$\tau = -\alpha \nu_t (\nabla \overline{\boldsymbol{U}} + \nabla \overline{\boldsymbol{U}}^T) + \frac{2}{3} k_{sgs} \boldsymbol{I}$$

$$\nu_t = C_\mu \frac{k_{sgs}^2}{\varepsilon_{sgs}} \left(\frac{k_{sgs}}{k_{sgs} + k_r} \right)$$

$$\frac{\partial k_{sgs}}{\partial t} + \nabla \cdot (k_{sgs} \bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k_{sgs} \right] + \alpha P - \varepsilon_{sgs}$$

$$\frac{\partial \varepsilon_{sgs}}{\partial t} + \nabla \cdot (\varepsilon_{sgs} \bar{\boldsymbol{U}}) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \nabla \varepsilon_{sgs} \right] + \frac{\varepsilon_{sgs}}{k_{sgs}} \left[C_{\varepsilon 1} P - C_{\varepsilon 2} \varepsilon_{sgs} \right]$$

$$P = \nu_t(\nabla\overline{\boldsymbol{U}} + \nabla\overline{\boldsymbol{U}}^T)\nabla\overline{\boldsymbol{U}}; \qquad C_{\varepsilon 2} = (11/6)f + (25/Re_T)f^2$$

$$Re_T = k_{sgs}^2/\nu\varepsilon_{sgs}; \quad f = (Re_T/30[\sqrt{1+60/Re_T} - 1]$$

$$\alpha = 1.5\left\{1 - C^*\left(\frac{k_{sgs}}{k_{sgs}+k_r}\right)^2\left[\left(\frac{\Delta x_i}{\sqrt{k_r}}\frac{\partial\sqrt{k_r}}{\partial x_i}\right)^2 + 0.11\right]^{-1}\right\}$$

$$\frac{\left(\Delta x_i \frac{\partial\sqrt{k_r}}{\partial x_i}\right)^2}{k_r} = \frac{\left\{\left(\Delta x\frac{\partial\sqrt{k_r}}{\partial x}\right)^2 + \left(\Delta y\frac{\partial\sqrt{k_r}}{\partial y}\right)^2 + \left(\Delta z\frac{\partial\sqrt{k_r}}{\partial z}\right)^2\right\}}{k_r}$$

$$C_\mu = 0.18; \quad C_{\varepsilon 1} = 1.55; \quad \sigma_k = 1.0; \quad \sigma_\varepsilon = 1.2; \quad C^* = 0.28$$

Warm-up exercise

Copy the entire Turbulence model directory into user directory:

```
of+ // reader may use OF1706+
mkdir -p $FOAM_RUN
foam
cp -r --parents src/TurbulenceModels $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/TurbulenceModels
```

Change the location of compiled files to relevant user directory:

```
find . -name Make
sed -i s/FOAM_LIBBIN/FOAM_USER_LIBBIN/g ./*/Make/files
./Allwmake
```

Copy the kEqn SGS model and change name to kEqnInagi:

```
cd turbulenceModels/LES
cp -r kEqn kEqnInagi
cd kEqnInagi
mv kEqn.H kEqnInagi.H
mv kEqn.C kEqnInagi.C
sed -i 's/kEqn/kEqnInagi/g *
sed -i 's/"OpenFoam Foundation"/"Your Name"/g *
```

Then (single line):

```
vi $WM_PROJECT_USER_DIR/src/TurbulenceModels/incompressible/
turbulentTransportModels/turbulentTransportModels.C
```

add following lines:

```
#include "kEqnInagi.H"
makeLESModel(kEqnInagi);
```

under lines for kEqn model.
Save and close the file, update InInclude directory and recompile:

```
wmakeLnInclude -u ../../../turbulenceModels
./../../../Allwmake
```

Then open `kEqnInagi.C` by:

```
vi kEqnInagi.C
```

replace

```
this->nut_ = Ck_*sqrt(k_)*this->delta();
```

with

```
dimensionedScalar verySmall
("verySmall",dimensionSet (0, 1, -1, 0, 0, 0, 0),VSMALL);
this->nut_ = 10*Ck_*k_*this->delta()/(10.0*sqrt(k_) +
this->delta()*sqrt(2*magSqr(symm(fvc::grad(this->U_)))) +
verySmall);
```

Then (touch single line)

```
touch ../../../incompressible/turbulentTransportModels/
turbulentTransportModels.C
./../../../Allwmake
```

Preparatory Work

Copy the kEqn SGS model and change name to kEpsilonSAS:

```
cd $WM_PROJECT_USER_DIR/src/TurbulenceModels
cd turbulenceModels/LES
cp -r kEqn kEpsilonSAS
cd kEpsilonSAS
mv kEqn.H kEpsilonSAS.H
mv kEqn.C kEpsilonSAS.C
sed -i 's/kqn/kEpsilonSAS/g *
sed -i 's/"OpenFoam Foundation"/"Your Name"/g *
```

Then (vi single line):

```
vi $WM_PROJECT_USER_DIR/src/TurbulenceModels/incompressible/
turbulentTransportModels/turbulentTransportModels.C
```

add following lines:

```
#include "kEpsilonSAS.H"
makeLESModel(kEpsilonSAS);
```

under lines for kEqn model.
Save and close the file, update lnInclude directory and recompile:

```
wmakeLnInclude -u ../../../turbulenceModels
./../../../Allwmake
```

H file

Under the list of `// Fields`:

add

```
volScalarField epsilon_;
volVectorField UMean_;
volScalarField kR_;
volScalarField dimlessGradkR_;
volScalarField alfa_;
volScalarField nutByAlfa_;
```

under

```
volScalarField k_;
```

Under the list of `// Model constants`:

replace

```
dimensionedScalar Ck_;
```

with

```
dimensionedScalar Cnu_;
dimensionedScalar Ce1_;
dimensionedScalar sigmaK_;
dimensionedScalar sigmaEps_;
```

Under the list of `// Protected Member Functions`:

add

```
virtual tmp<fvScalarMatrix> epsilonSource() const;
```

under

```
virtual tmp<fvScalarMatrix> kSource() const;
```

Under the list of `// Member Functions`:

replace

```
//- Return sub-grid disipation rate
virtual tmp<volScalarField> epsilon() const;

//- Return the effective diffusivity for k
tmp<volScalarField> DkEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DkEff",
        this->nut_ + this->nu())
    );
}
```

with (continue to next page...)

```
//- Return sub-grid disipation rate
virtual tmp<volScalarField> epsilon() const
{
    return epsilon_;
}
//- Return the effective diffusivity for k
tmp<volScalarField> DkEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DkEff",
        (nutByAlfa_ + this->nu())/sigmaK_)
    );
}
```

```
//- Return the effective diffusivity for epsilon
tmp<volScalarField> DepsilonEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DepsilonEff",
        (this->nutByAlfa_ + this->nu())/sigmaEps_)
    );
}
```

C file

Under the list of `// Protected Member Functions`:

within the function of

```
template<class BasicTurbulenceModel>
void kEpsilonSAS<BasicTurbulenceModel>::correctNut()
```

replace

```
    this->nut_ = Ck_*sqrt(k_)*this->delta();
```

with (to next page, and next, and next...)

```
kR_ = 0.5*magSqr(UMean_ - this->U_);
volVectorField gradSqrtkR = fvc::grad(sqrt(kR_));

surfaceVectorField fV = this->mesh_.Sf();
volScalarField surfaceSumX =
0.5*fvc::surfaceSum(mag(fV.component(0)));
volScalarField surfaceSumY =
0.5*fvc::surfaceSum(mag(fV.component(1)));
volScalarField surfaceSumZ =
0.5*fvc::surfaceSum(mag(fV.component(2)));
```

```
volScalarField cv
(
        IOobject
        (
                "cv",
                this->runTime_.timeName(),
                this->mesh_,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
        ),
        this->mesh_,
        dimensionedScalar("zero",dimVolume,0.0)
);

cv.ref() = this->mesh_.V();
```

```
dimensionedScalar surfaceMin
("surfaceMin",dimensionSet (0, 2, 0, 0, 0, 0, 0),VSMALL);

dimlessGradkR_ =
(
 sqr(cv / (surfaceSumX + surfaceMin) *
 gradSqrtkR.component(0)) +
 sqr(cv / (surfaceSumY + surfaceMin) *
 gradSqrtkR.component(1)) +
 sqr(cv / (surfaceSumZ + surfaceMin) *
 gradSqrtkR.component(2))
)
/ (kR_ + this->kMin_);
```

```
alfa_ =
1.5*
   (
     1.0 -
     0.28 * sqr(k_ / (k_ + kR_ + this->kMin_)) /
     (dimlessGradkR_ + 0.11)
   );

// Calculate SGS nut
nutByAlfa_ = Cnu_*sqr(k_)/(epsilon_+this->epsilonMin_)*
(k_/(k_+kR_+this->kMin_));
this->nut_ = alfa_*nutByAlfa_;
```

also under

```
template<class BasicTurbulenceModel>
tmp<fvScalarMatrix> kEqn<BasicTurbulenceModel>::kSource() cons
{
    return tmp<fvScalarMatrix>
    (
        new fvScalarMatrix
        (
            k_,
            dimVolume*this->rho_.dimensions()*k_.dimensions()
            /dimTime
        )
    );
}
```

add

```
template<class BasicTurbulenceModel>
tmp<fvScalarMatrix> kEpsilonSAS<BasicTurbulenceModel>::
epsilonSource() const
{
    return tmp<fvScalarMatrix>
    (
        new fvScalarMatrix
        (
            epsilon_,
            dimVolume*this->rho_.dimensions()*
            epsilon_.dimensions()/dimTime
        )
    );
}
```

Within the `Constructors`:

replace

```
Ck_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Ck",
        this->coeffDict_,
        0.094
    )
)
```

with (the next page, and next, and next...)

```
epsilon_
(
    IOobject
    (
        IOobject::groupName("epsilon", this->U_.group()),
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_
),
```

```
UMean_
(
    IOobject
    (
        IOobject::groupName("UMean", this->U_.group()),
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    ),
    this->mesh_
),
```

```
kR_
(
    IOobject
    (
        IOobject::groupName("kR", this->U_.group()),
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_,
    dimensionedScalar("kR",this->k_.dimensions(),SMALL)
),
```

```
dimlessGradkR_
(
        IOobject
        (
                "dimlessGradkR",
                this->runTime_.timeName(),
                this->mesh_,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
        ),
        this->mesh_,
        dimensionedScalar("dimlessGradkR",dimless,SMALL)
),
```

```
alfa_
(
    IOobject
    (
        IOobject::groupName("alfa", this->U_.group()),
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_,
    dimensionedScalar("alfa",dimless,SMALL)
),
```

```
nutByAlfa_
(
    IOobject
    (
        IOobject::groupName("nutByAlfa",
        this->U_.group()),
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_,
    dimensionedScalar("nutByAlfa",
    this->nut_.dimensions(),SMALL)
),
```

```
Cnu_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cnu",
        this->coeffDict_,
        0.18
    )
),
```

```
Ce1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Ce1",
        this->coeffDict_,
        1.55
    )
),
```

```
sigmaK_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmaK",
        this->coeffDict_,
        1.0
    )
),
```

```
sigmaEps_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmaEps",
        this->coeffDict_,
        1.2
    )
)
```

Also under

```
bound(k_, this->kMin_);
```

add

```
bound(epsilon_, this->epsilonMin_);
```

Within the function of

```
template<class BasicTurbulenceModel>
bool kEqn<BasicTurbulenceModel>::read()
```

replace

```
Ck_.readIfPresent(this->coeffDict());
```

with

```
Cnu_.readIfPresent(this->coeffDict());
Ce1_.readIfPresent(this->coeffDict());
sigmaK_.readIfPresent(this->coeffDict());
sigmaEps_.readIfPresent(this->coeffDict());
```

Remove the function below:

```
template<class BasicTurbulenceModel>
tmp<volScalarField> kEpsilonSAS<BasicTurbulenceModel>::
epsilon() const {
    return tmp<volScalarField>
    (
        new volScalarField
        (   IOobject
            (
                IOobject::groupName("epsilon", this->U_.group(
                this->runTime_.timeName(),
                this->mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),this->Ce_*k()*sqrt(k())/this->delta()));}
```

Within the function of:

```
template<class BasicTurbulenceModel>
void kEpsilonSAS<BasicTurbulenceModel>::correct()
```

before

```
    // Local references
```

add

```
    Info << "This is kEpsilonSAS" << endl;
```

Also before

```
    tmp<fvScalarMatrix> kEqn
```

add following:

```
// Calculate Ce2 dynamically
tmp<volScalarField> ReT(sqr(k_)/(this->nu()*
(epsilon_+this->epsilonMin_)));
tmp<volScalarField> f(sqrt(sqr(ReT()/30) +
ReT()/15)-ReT()/30);
volScalarField Ce2(11/6*f()+25/(ReT()+SMALL)*sqr(f()));
ReT.clear();
f.clear();

volScalarField epsilon(epsilon_ + this->epsilonMin_);
volScalarField k(k_ + this->kMin_);
```

```
// Dissipation equation
tmp<fvScalarMatrix> epsEqn
(
    fvm::ddt(alpha, rho, epsilon_)
  + fvm::div(alphaRhoPhi, epsilon_)
  - fvm::laplacian(alpha*rho*DepsilonEff(), epsilon_)
 ==
    Ce1_*alpha*rho*G*epsilon_/k
  - fvm::SuSp(((2.0/3.0)*Ce1_)*alpha*rho*divU, epsilon_)
  - fvm::Sp(Ce2*alpha*rho*epsilon_/k, epsilon_)
  + epsilonSource()
  + fvOptions(alpha, rho, epsilon_)
);
```

```
epsEqn.ref().relax();
fvOptions.constrain(epsEqn.ref());
//epsEqn.ref().boundaryManipulate
(epsilon_.boundaryFieldRef());
solve(epsEqn);
fvOptions.correct(epsilon_);
bound(epsilon_, this->epsilonMin_);
```

Within the function of

```
tmp<fvScalarMatrix> kEqn
```

replace

```
alpha*rho*G
```

with

```
alpha*rho*G*alfa_
```

also replace

```
- fvm::Sp(this->Ce_*alpha*rho*sqrt(k_)/this->delta(), k_)
```

with

```
- fvm::Sp(alpha*rho*epsilon_/k, k_)
```

k-equation with Inagi wall damping function

Copy the pitzDaily tutorial to run directory (cp one line):

```
run
rm -r pitzDaily
cp -r  $FOAM_TUTORIALS/incompressible/pisoFoam/LES/
pitzDaily $FOAM_RUN/pitzDailyKInagi
cd $FOAM_RUN/pitzDailyKInagi
```

In `/system/controlDict`:

change `endTime` to 0.3, and comment out all functionObject except `Umean`.

In `/constant/turbulenceProperties`:

replace

```
    LESModel        dynamicKEqn;
```

with

```
    LESModel        kEqInagi;
```

Then run

```
blockMesh
pisoFoam >& log&
```

Getting Started

Copy pitzDaily tutorial to run directory:

```
run
rm -r pitzDaily
cp -r  $FOAM_TUTORIALS/incompressible/pisoFoam/LES/pitzDaily $
cd $FOAM_RUN/pitzDailyKESAS
```

Remove `nuTilda` and `s` in `/0`, then:

```
cp 0/k 0/epsilon
cp 0/U 0/UMean
```

followed by changing `object` to relevant fields in `0/epsilon` and `0/UMean`. Also change the remaining content in `0/epsilon` as:

```
dimensions      [0 2 -3 0 0 0 0];
internalField   uniform 79e-5;
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 79e-5;
    }
    outlet
    {
        type            zeroGradient;
    }
```

```
    upperWall
    {
        type            fixedValue;
        value           uniform 0;
    }
    lowerWall
    {
        type            fixedValue;
        value           uniform 0;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

/system directory

In `/system/controlDict`:

change `endTime` to 0.3, and comment out all functionObject except `UMean`.

In `/system/fvSchemes`:

add

```
div(phi,epsilon)      Gauss limitedLinear 1;
```

under

```
div(phi,k)      Gauss limitedLinear 1;
```

In `/system/fvSolution`:

replace

```
"(U|k|B|nuTilda|s)"
```

with

```
"(U|k|epsilon|B|nuTilda|s)"
```

/constant directory

Keep the /constant/transportProperties unchanged.

In /constant/turbulenceProperties:
replace

```
LESModel        dynamicKEqn;
```

with

```
LESModel        kEpsilonSAS;
```

also replace

```
delta       cubeRootVol;
```

with

```
delta       vanDriest;
```

Run the case

Run commands:

```
blockMesh
pisoFoam >& log&
```

The inlet velocity are set to be $U = 10m/s$ with turbulence intensity of 2%, 1% and 1% at x, y and z direction respectively.

Initial condition for pressure, p, eddy viscosity, ν_t, k and ε value are kept as close as in the original tutorial cases. Result are analysed at $t = 0.3s$.

Divergence!!!

You will probably find the computation diverges.

Please go back to `kEpsilonSAS.C` file, in the k-equation:

replace

```
alpha*rho*G*alfa_
```

with

```
alpha*rho*G//*alfa_
```

Then recompile and run again.

Velocity Magnitude

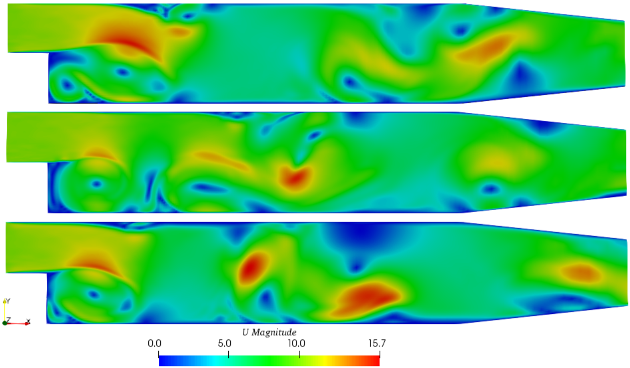

Figure: Instantaneous velocity magnitude profile for k-equation SGS model with Inagi wall damping function (top), k-ε SGS model (middle) and k-equation SGS model with normal Δ.

Time-averaged Velocity

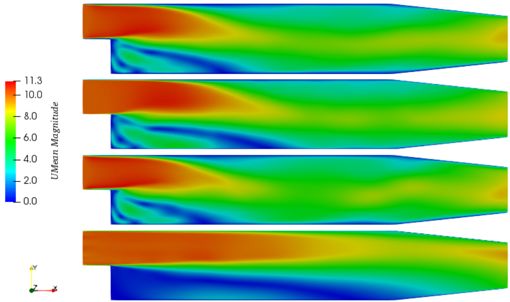

Figure: Time-averaged velocity profile for k-equation SGS model with Inagi wall damping function (top), k-ε SGS model (second from top) and k-equation SGS model with normal Δ (third from top), and k-ε RANS model (bottom).

Conclusion and Future Work

- Inagi wall damping function and new Two-equation SGS model have been implemented;
- Preliminary test shows the new model can lead to divergence;
- The numerical instability can be improved by removing the energy backscatter term in k-equation for the new model;
- The new model (without the energy backscatter term) can capture the unsteadiness;
- The results of k-equation SGS model with Inagi wall damping function shows difference from the one with normal Δ.
- The cause of divergence is subject to further investigation;
- After improving the model's numerical stability, an extensive test will be down based on channel flow, backward facing step and even more practical scenarios.

Thank you for your attention

Good luck with a successful convergence!