A | Appendix A

In this appendix, the C++ code implementation for the Algebraic turbulent viscosity model and the functionObject strain rate stress tensor is presented. In addition, the Python script used for coarsening is also presented. The series of Python scripts used for the 3D preprocessing are not shown given its excessive lengths.

A.0.1. Zero Equation Turbulent Viscosity

To implement a custom turbulence model, an existing reference model is typically used as a starting point. In our case, the `kEpsilon` turbulence model was selected as the base. Since the `kEpsilon` model falls under the RANS category, the new turbulence model will also be defined as a RANS model. The custom implementation consists of a `.C` source file and a corresponding `.H` header file, both of which are presented below.

1. Create a new folder named `mixingLengthModelFINAL` in the following directory:

   ```
   /turbulenceModels/RAS/
   ```

   and place the `mixingLengthModelFINAL.C` and `mixingLengthModelFINAL.H` source files inside it.

2. Edit the `Make/files` file with the line `RAS/mixingLengthModelFINAL/mixingLengthModelFINA` and the `Make/options` with :

   ```
   EXE_INC = \-I$(FOAM_SRC)/TurbulenceModels/incompressible/RAS/lnInclude \-
       I$(FOAM_SRC)/transportModels/incompressible/lnInclude
   LIB_LIBS = \-lincompressibleTransportModels \
   -lincompressibleTurbulenceModels
   ```

3. Edit the file `/TurbulenceModels/incompressible/turbulenceModels.C` and register the new model by adding:

   ```
   #include "mixingLengthModelFINAL.H"
   makeRASModel(mixingLengthModelFINAL);
   ```

4. Compile the `Make` by running in the `turbulenceModels` directory, this registered the turbulence model, and to use it in incompressible simulation the compile the `Make` in the `incompressibleFolder`:

```
cd $FOAM_SRC/TurbulenceModels/incompressible/RAS/mixingLengthModel
wmake
```

5. To use the model in your simulation case, modify the `constant/turbulenceProperties` file as follows:

```
RAS
{
    RASModel mixingLengthModel;
    turbulence on;
    printCoeffs on;
    mixingLengthModelCoeffs
    {
        L 0.01 meters as an example value
    }
}
```

mixingLengthModelFINAL.C

```
#include "mixingLengthModelFINAL.H"
#include "fvOptions.H"
#include "bound.H"
namespace Foam
{
namespace RASModels
{
// --- Dummy implementation for k() so that the class is concrete ---
template<class BasicTurbulenceModel>
tmp<volScalarField> mixingLengthModelFINAL<BasicTurbulenceModel>::k() const
{
    return tmp<volScalarField>(
        new volScalarField
        (
            IOobject
            (
                "k",
                this->runTime_.timeName(),
```